МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Кваліфікаційна наукова
праця на правах рукопису

УГРИНОВСЬКИЙ БОГДАН ВОЛОДИМИРОВИЧ

УДК 004.052

ДИСЕРТАЦІЯ

МЕТОДИ І ЗАСОБИ ПІДВИЩЕННЯ НАДІЙНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ З УРАХУВАННЯМ ПРОЦЕСУ ЙОГО СТАРІННЯ

121 – «Інженерія програмного забезпечення»
12 – «Інформаційні технології»

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Б.В. Угриновський

Науковий керівник:
Яковина Віталій Степанович
доктор технічних наук, професор

Львів - 2022

АНОТАЦІЯ

Угриновський Б. В. Методи і засоби підвищення надійності програмного забезпечення з урахуванням процесу його старіння. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 «Програмна інженерія» (12 – «Інформаційні технології»). – Національний університет «Львівська політехніка», Львів, 2022.

У дисертаційній роботі розв'язано актуальну науково-прикладну задачу у галузі програмної інженерії – підвищення рівня надійності програмного забезпечення мобільних систем шляхом визначення факторів, що впливають на процес його старіння, побудови математичних моделей старіння і омолодження програмного забезпечення та розроблення відповідних засобів.

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку літературних джерел та додатків.

У **вступі** обґрунтовано актуальність теми дисертаційного дослідження; описано зв’язок роботи з науковими програмами, планами, темами; сформульовано мету та основні завдання дисертаційної роботи; представлено методи дослідження та визначено наукову новизну та відображено практичне значення одержаних результатів дослідження; презентовано списки опублікованих праць за тематикою дисертаційної роботи та конференцій, на котрих було апробовано основні результати дисертаційної роботи.

В **першому розділі** розглянуто поняття старіння програмного забезпечення та такі основні характеристики як дефекти, помилки, ефекти, метрики та фактори старіння. Визначено, що явище старіння має значний негативний вплив на надійність та продуктивність програмного забезпечення. Розглянуто та проаналізовано різні методи дослідження та моделювання явища старіння програмного забезпечення. Для теоретичного опису процесу старіння і омолодження програмного забезпечення використовуються аналітичні моделі на основі ланцюгів Маркова з неперервним часом розподілу. В свою чергу, гібридні підходи дослідження та опису процесу старіння програмного забезпечення є перспективним напрямком у вивченні старіння,

оскільки включають переваги аналітичних моделей та методів на основі вимірювань метрик старіння. Враховуючи загальну складність програмних систем і особливості процесу старіння, ефективним інструментом протидії явищу старіння є виконання процедури омолодження програмного забезпечення. Встановлено, що урахування «теплого» та «холодного» механізмів процедури омолодження програмного забезпечення в побудові моделей старіння та методів омолодження є важливим для покращення користувацького досвіду, мінімізації часу простою системи та максимізації ефективності процедури омолодження загалом. Обґрунтовано важливість емпіричного та теоретичного дослідження процесу старіння та методів протидії його ефектам в мобільних операційних системах, зокрема, Android, оскільки, мобільні та вбудовані пристрої вразливі до ефектів старіння, а їх повсякденне використання вимагає забезпечення високого рівня надійності. Сформульовано актуальні науково-прикладні задачі, які будуть розв’язуватись в даній роботі.

У **другому розділі** виконано аналіз існуючої моделі процесу старіння та омолодження програмного забезпечення для операційної системи Android з урахуванням активності використання мобільного пристрою користувачем, визначено основні недоліки цієї моделі та запропоновано покращені моделі старіння та омолодження, які враховують як активність використання мобільного пристрою користувачем, так і механізми «теплого» та «холодного» виконання процедури омолодження, різні рівні старіння та можливі стратегії виконання процедури омолодження, а також, фактор рівня заряду батареї. Для математичного опису процесу старіння та омолодження програмного забезпечення використано апарат ланцюгів Маркова з неперервним часом розподілу. Запропоновані моделі старіння та омолодження можуть бути застосовані для оцінки таких показників якості існуючого програмного забезпечення як надійність та продуктивність, дозволяють отримувати вирази для показників ефективності омолодження програмного забезпечення, а також, можуть використовуватись для проектування та вибору параметрів методу омолодження програмного забезпечення, що дає змогу формувати технічне завдання на його розроблення.

Виконані симуляції процесу старіння та омолодження з допомогою моделі з урахуванням різних рівнів старіння програмного забезпечення та стратегій омолодження показали, що ефективною стратегією є повторюване виконання процедури омолодження в стані «старіння», яке дозволяє збільшити ймовірність перебування системи в «молодому» стані під час її активного використання користувачем в 1,37-1,77 рази порівняно із симуляцією без виконання омолодження. Практичне значення цієї моделі в тому, що використання різних рівнів дозволяє застосовувати методи на основі вимірювань та порогів метрик старіння для визначення стану системи та формування статистики.

Аналіз запропонованої моделі з урахуванням «теплого» та «холодного» механізму омолодження показав, що вона може бути використана для вибору оптимального механізму для тих чи інших умов і матиме вагомий вплив на характеристики досвіду користувача та надійності.

Моделювання процесу старіння та омолодження з урахуванням фактору заряду батареї показало, що рівень заряду може мати вагомий вплив на результат прогнозу часу виконання омолодження програмного забезпечення. Зокрема, без урахування часу до повного розряду батареї заплановане омолодження може не відбутись, що було підтверджено виконаними симуляціями, де модель з урахуванням заряду батареї показувала оптимальний час виконання омолодження в 4,5 рази менший, ніж аналогічний прогноз без урахування того факту, що батарея розрядиться швидше, ніж відбудеться омолодження.

В **третьому розділі** представлено результати експериментальних досліджень явища старіння програмного забезпечення в операційній системі Android, зокрема, виконано аналіз метрик та факторів старіння.

Запропоновано та експериментально перевірено дві метрики графічного інтерфейсу користувача, нові в контексті старіння програмного забезпечення: тривалість відображення кадрів та кількість «зіпсованих» кадрів. Метрика тривалості відображення кадрів може бути використана у більшій кількості реальних сценаріїв використання програмного забезпечення, ніж метрика тривалості запуску Android Activity, і дозволяє отримувати неперервні часові ряди даних. Зокрема, у виконаних

експериментах метрика тривалості відображення кадрів дозволяє отримати в 4 рази більше записів, ніж метрика тривалості запуску Android Activity.

Досліджено нові фактори старіння, а саме, крос-платформові застосунки на базі фреймворку Flutter та сценарій використання мобільного пристрою із паузами під час генерування робочого навантаження. Дослідження показало, що розглянуті фактори мають значний вплив на вимірювані метрики відображення кадрів користувацького інтерфейсу, зокрема, тип користувацьких застосунків має більший вплив, ніж сценарій використання. Результати показують, що крос-платформові застосунки Flutter є більш вразливими до ефектів старіння, ніж стандартні native застосунки.

Визначено, що системні процеси system_server, com.android.systemui та surfaceflinger є найбільш вразливі до ефектів старіння, тому можуть бути використані в методах омолодження програмного забезпечення в якості індикаторів старіння в операційній системі. Крім того, в залежності від сценарію використання мобільного пристрою користувачем старіння може спостерігатись і у інших сервісах, наприклад, cameraserver та audioserver.

Аналіз користувацьких застосунків показав, що для протидії ефектам старіння на їх рівні важливо вживати заходів та розробляти відповідні засоби для попередження виникнення помилок старіння, а саме, попередження помилок з переповненням пам’яті, мінімізація навантаження на потік відповідальний за відображення користувацького інтерфейсу, щоб зменшити кількість затримок кадрів.

На основі виконаних теоретичних та практичних досліджень запропоновано і описано метод омолодження програмного забезпечення для операційної системи Android, який використовує комплексну модель, описану в другому розділі, для прогнозування часу виконання процедури омолодження. Метод, так само як і модель, визначає поступові рівні старіння системи, дозволяє будувати різні стратегії омолодження, враховує різні механізми процедури омолодження та враховує такі фактори як активність використання мобільного пристрою користувачем та рівень заряду батареї.

У **четвертому розділі** описано відому методологію дослідження старіння програмного забезпечення для операційної системи Android, яка використана в даній

роботі для дослідження метрик, факторів та особливостей процесів старіння. Методологія складається із шести етапів: визначення стратегії моніторингу, визначення алгоритму генерування робочого навантаження, планування стресових тестів, виконання стресових тестів, аналіз зібраних даних та представлення результатів експерименту.

Розроблено фреймворк для виконання стресового тестування мобільних застосунків ОС Android та виконання аналізу зібраних даних у відповідності до попередньо визначеної методології дослідження. Фреймворк складається із трьох взаємозалежних модулів: модуль виконання стресових тестів; модуль генерування часових рядів метрик; модуль аналізу часових рядів.

Розроблено структуру модулів програмного засобу омолодження програмного забезпечення для операційної системи Android, в контексті якої можливо реалізувати запропонований в третьому розділі метод омолодження програмного забезпечення. Визначено основні вимоги до розроблюваного комплексу у вигляді системного сервісу, що працюватиме у фоновому режимі. Крім того, запропоновано засоби превентивної протидії старінню та омолодження на рівні користувацьких застосувань шляхом використання таких методів системного сервісу старіння та омолодження як функція для отримання даних про стан старіння системи та функція зворотного виклику на подію про виявлення старіння в системі.

Ключові слова: старіння програмного забезпечення, омолодження програмного забезпечення, надійність програмного забезпечення, досвід користувача, операційна система, Android, метрики старіння, модель старіння та омолодження, дефект старіння, помилка старіння, відмова старіння, фактор старіння, час до відмови старіння, час до виснаження ресурсу, ланцюг Маркова з неперервним часом розподілу.

ABSTRACT

Uhrynovskyi B.V. Methods and tools of software reliability improvement considering software aging. – Qualifying scientific work on the rights of the manuscript.

Thesis paper for achievement of the scientific degree Doctor of Philosophy in the specialty 121 “Software Engineering” (12 – “Information Technology”). – Lviv Polytechnic National University, Lviv, 2022.

The dissertation solves a topical scientific and applied problem in the field of software engineering - increasing the software reliability for mobile systems by identifying factors influencing the aging process, building software aging and rejuvenation mathematical models and developing appropriate tools.

The dissertation consists of an introduction, four chapters, conclusions, a list of references and appendices.

The introduction substantiates the relevance of the topic of dissertation research; describes the connection of work with scientific programs, plans, topics; the purpose and main tasks of the dissertation are formulated; research methods are presented and scientific novelty is determined and the practical significance of the obtained research results is reflected; lists of published works on the topic of dissertation work and conferences were presented, at which the main results of the dissertation work were tested.

The first chapter discusses the concept of software aging and key characteristics such as defects, errors, effects, metrics and aging factors. It is determined that the phenomenon of aging has a significant negative impact on the software reliability and performance. Various methods of research and modeling of the software aging phenomenon are considered and analyzed. Analytical models based on Continuous-Time Markov chains are used for the theoretical description of the software aging and rejuvenation process. In turn, studying of hybrid approaches of software aging process research is a promising area, as they include the advantages of analytical models and measurements-based methods. Given the general complexity of software systems and the peculiarities of the aging process, an effective tool to combat the aging phenomenon is to perform a procedure of software rejuvenation. It has been found that considering the warm and cold software rejuvenation procedure mechanisms in building of aging models and rejuvenation methods is important

to improve the user experience, minimize system downtime and maximize the effectiveness of the rejuvenation procedure in general. The importance of empirical and theoretical research of the aging process and methods of counteracting its effects in mobile operating systems, in particular, Android, is justified, because mobile and embedded devices are vulnerable to the effects of aging, and their daily use requires a high level of reliability. The actual scientific and applied problems that will be solved in this work are formulated.

The second chapter analyzes the existing model of software aging and rejuvenation for the Android operating system considering the usage activity of mobile device, identifies the main disadvantages of this model and offers improved models of aging and rejuvenation, which take into account both user activity and mechanisms of warm and cold rejuvenation, different aging levels and possible rejuvenation strategies, as well as the battery charge factor. For the mathematical description of the software aging and rejuvenation process, a Continuous-Time Markov chain apparatus was used. The proposed aging and rejuvenation models can be used to assess the quality of existing software, allow to obtain expressions for software rejuvenation efficiency indicators, and can be used to design software rejuvenation means and select their parameters, which allows to form a technical task for its development.

Simulations of the aging and rejuvenation process using the model considering different aging levels and rejuvenation strategies have shown that an effective strategy is to repeat the rejuvenation procedure in the state of “aging”, which increases likelihood of the system in a “young” state during its active user use in 1.37-1.77 times compared to the simulation without rejuvenation. The practical significance of this model is that the use of different levels allows the use of measurements- and threshold-based methods to determine the state of the system and the formation of statistics.

Analysis of the proposed model considering the warm and cold rejuvenation mechanism showed that it can be used to select the optimal mechanism for certain conditions and will have a significant impact on the user experience and software reliability.

Modeling of the aging and rejuvenation process considering the battery charge factor has shown that the charge level can have a significant impact on the result of the forecast of the software rejuvenation time. In particular, the planned rejuvenation may not take place

without taking into account the time until the battery is fully discharged, which was confirmed by simulations where the battery charge model showed the optimal rejuvenation time 4.5 times less than the same forecast without taking into account the fact that the battery will discharge faster than rejuvenation.

The third chapter presents the results of experimental studies of the software aging phenomenon in the Android operating system, in particular, the analysis of metrics and aging factors.

Two new graphical user interface metrics have been proposed and experimentally tested in the context of software aging: frame draw time and junky frames count. Frame draw time metric can be used in more real-world software usage scenarios than the Android Activity launch time metric and allows to get continuous time series. In particular, performed experiments allow to get 4 times more frame draw time metric records than the Android Activity launch time metric.

New aging factors have been explored, namely cross-platform applications based on the Flutter framework and the scenario of using a mobile device with pauses when generating workload. The study showed that the considered factors have a significant impact on the measured user interface metrics, in particular, the type of user applications has a greater impact than the usage scenario. The results show that Flutter cross-platform applications are more vulnerable to the effects of aging than standard native applications.

It is determined that the system processes system_server, com.android.systemui and surfaceflinger are the most vulnerable to the aging-related effects, so they can be used in software rejuvenation methods as indicators of aging in the operating system. In addition, depending on the user's use of the mobile device, aging may be observed in other services, such as cameraserver and audioserver.

Analysis of user applications has shown that it is important to take measures and develop appropriate tools to prevent aging errors, namely, prevent errors with memory overflow, minimize the load on the thread responsible for displaying the user interface to reduce the number of frame delays.

Based on the performed theoretical and practical research, the software rejuvenation method for the Android operating system is proposed and described. The method uses the

complex model described in the second chapter to predict the time of the rejuvenation procedure. The method, like the model, determines the gradual levels of system aging, allows the building of different rejuvenation strategies, takes into account different mechanisms of the rejuvenation procedure, and takes into account factors such as mobile device user activity and battery charge.

The fourth chapter describes the general methodology for researching software aging in the Android operating system, which is used in this paper to study the metrics, factors, and features of aging processes. The methodology consists of six stages: determining the monitoring strategy, determining the algorithm for workload generation, stress tests planning, stress tests performing, analyzing the collected data, and presenting the results of the experiment.

A framework has been developed to perform stress testing of Android mobile applications and perform analysis of collected data in accordance with a predefined research methodology. The framework consists of three interdependent modules: a stress test module; metrics time-series generation module; time-series analysis module.

The structure of software modules for the Android operating system has been developed, in the context of which it is possible to implement the software rejuvenation method proposed in the third chapter. The main requirements to the developed rejuvenation complex are determined, in particular, it can be implemented in the form of a system service that will run in the background. In addition, tools are proposed to prevent aging at the user applications level by using system service methods such as a function to obtain system aging status and a callback function for the system aging event.

Keywords: software aging, software rejuvenation, software reliability, user experience, operating system, Android, aging-related metrics, aging and rejuvenation model, aging-related defect, aging-related error, aging-related failure, aging-related factor, time to aging failure, time to resource consumption, Continuous Time Markov chain.

СПИСОК ОПУБЛІКОВАНИХ ПРАЦЬ ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковано основні наукові результати дисертації

1. Яковина В.С., Угриновський Б.В. Старіння програмного забезпечення в контексті його надійності: огляд проблематики. Науковий вісник НЛТУ України. 2019. 29(5), 123-128. DOI: https://doi.org/10.15421/40290525

2. Яковина В.С., Угриновський Б.В. Старіння програмного забезпечення мобільних додатків: огляд проблематики. Науковий вісник НЛТУ України. 2020. 30(2), 107-112. DOI: https://doi.org/10.36930/40300219

3. Яковина В.С., Угриновський Б.В. Метрики інтерфейсу користувача для виявлення явища старіння програмного забезпечення в мобільній системі Android. Вісник НУЛП «Інформаційні системи та мережі». 2021. 9, 32-43. DOI: https://doi.org/10.23939/sisn2021.09.032

4. Яковина В.С., Угриновський Б.В. Дослідження системних процесів та користувацьких додатків операційної системи Android в контексті старіння програмного забезпечення. Вісник ХНУ: Технічні науки. 2021. 295(2), 64-70. DOI: https://www.doi.org/10.31891/2307-5732-2021-295-2-64-70

5. Yakovyna, V. S., Uhrynovskyi, B. V. Android software aging and rejuvenation model considering the battery charge. Radio Electronics, Computer Science, Control. 2021. (4), 140-148. DOI: https://doi.org/10.15588/1607-3274-2021-4-13

6. Yakovyna V. S., Uhrynovskyi B. V. Extended software aging and rejuvenation model for Android operating system considering different aging levels and rejuvenation procedure types. Computer systems and information technologies. 2021. 3, 116-124. DOI: https://doi.org/10.31891/CSIT-2021-5-9

7. Яковина В.С., Угриновський Б.В. Метод омолодження програмного забезпечення для операційної системи Android з використанням комплексної моделі його старіння на підставі ланцюга Маркова. Науковий вісник НЛТУ України. 2021. 31(6), 97-103. DOI: https://doi.org/10.36930/40310615

Праці, які засвідчують апробацію матеріалів дисертації

1. Yakovyna V. S., Uhrynovskyi B. V., Bachkay O. Software Failures Forecasting by Holt - Winters, ARIMA and NNAR Methods. *IEEE 14th International Conference on*

Computer Sciences and Information Technologies. 2019. DOI: https://doi.org/10.1109/STC-CSIT.2019.8929863

2. Yakovyna V. S., Uhrynovskyi B. V. User-Perceived Response Metrics in Android OS for Software Aging detection. *IEEE 15th International Conference on Computer Sciences and Information Technologies*. 2020. DOI: https://doi.org/10.1109/CSIT49958.2020.9322031

3. Яковина В.С., Угриновський Б.В. Модель старіння та омолодження програмного забезпечення для платформи Android. *XX Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2020. 5-8 червня.

4. Яковина В.С., Угриновський Б.В. Засоби протидії явищу старіння програмного забезпечення в операційній системі Android. *XXI Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2021. 3-7 червня.

5. Yakovyna, V. S., Uhrynovskyi, B. V. Aging of Native and Flutter Applications in Android OS in Various Usage Scenarios. *IEEE 16th International Conference on Computer Sciences and Information Technologies.* 2021. DOI: https://doi.org/10.1109/CSIT52700.2021.9648777

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ARIMA – (англ. Autoregressive Integrated Moving Average), інтегрована модель авторегресії;

ALT – (англ. Activity Launch Time), тривалість запуску активності;

CTMC – (англ. Continuous-Time Markov Chain), ланцюг Маркова з неперервним часом розподілу;

DBN – (англ. Deep Belief Network), глибинна мережа переконань;

DSPN – (англ. Deterministic and Stochastic Petri Nets), детерміністичні та стохастичні мережі Петрі;

FDT – (англ. Frame Draw Time), тривалість відображення кадру;

FPS – (англ. Frames Per Seconds), кадри за секунду;

GUI – (англ. Graphical User Interface), графічний інтерфейс користувача;

ID3 – (англ. Decision Tree), дерево рішень;

JFC – (англ. Junky Frames Count), кількість «зіпсованих» кадрів;

JFR – (англ. Junky Frames Ratio), частка зіпсованих кадрів;

PSS – (англ. Proportional Set Size), пропорційний розмір набору;

RSS – (англ. Resident Set Size), резидентний розмір набору;

KSD – (англ. Kernel Swap Daemon), демон підкачки ядра;

LMKD – (англ. Low Memory Killer Daemon), демон-вбивця низького рівня пам’яті;

PCA – (англ. Principal Component Analysis), Аналіз основних компонентів;

POMDP – (англ. Partially Observable Markov Decision Processes), частково спостережувані процеси прийняття рішень Маркова;

MDP – (англ. Markov Decision Process), процес прийняття рішень Маркова;

MRGP – (англ. Markov regenerative process), Марковські регенераційні процеси;

TTAF – (англ. Time to Aging Failure), час до відмови старіння;

TTRE – (англ. Time to Resource Exhaustion), час до виснаження ресурсу;

UI – (англ. User Interface), інтерфейс користувача;

UX – (англ. User Experience), досвід користувача;

SPN – (англ. Stochastic Petri Net), стохастична мережа Петрі;

SRN – (англ. Stochastic Reward Nets), стохастичні мережі нагородження;

SVM – (англ. Support Vector Machine), метод опорних векторів;

АЗ – апаратне забезпечення;

ОС – операційна система;

ПЗ – програмне забезпечення;

ПК – персональний комп’ютер.

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі глобалізації та Інтернету речей ПЗ стало одним з найважливіших елементів для забезпечення роботи компаній та організацій у їхньому бізнес-середовищі. Разом з тим, ПЗ використовується у повсякденному житті звичайними людьми в ПК, мобільних пристроях, наручних годинниках, тощо. Потреби користувачів ПЗ та сфери його застосування часто можуть бути критично важливими для життя і здоров'я людини, наприклад: сфери банківських операцій, здоров'я та лікування, державні послуги. З огляду на це, такі характеристики якості ПЗ як надійність та продуктивність є критичними властивостями більшості сучасної техніки в цілому і ПЗ зокрема. Таким чином, питання підвищення рівня надійності та продуктивності є ключовими для сучасного етапу розвитку програмно-апаратних систем.

Стрімкий розвиток технологій сприяє значному збільшенню складності ПЗ, що в свою чергу збільшує імовірність виникнення помилок в ПЗ. Будь-яке ПЗ – це система, яка являє собою сукупність взаємодіючих елементів в певному середовищі. Кожен елемент системи може містити різного роду дефекти, які можуть призводити до помилок роботи ПЗ. Чим складніша система, тобто, чим більше в ній елементів, тим більше можливих комбінацій дефектів, які спричиняють виникнення помилок. В таких умовах існує категорія дефектів ПЗ, які важко ізолювати та дорого виправляти на етапі розроблення. Важливою особливістю дефектів, які важко ізолювати, є те, що вони не стають причинами відмов відразу, а накопичуються в стані системи із моменту початку роботи системи, що спричиняє поступове погіршення продуктивності роботи ПЗ і тільки через певний випадковий інтервал часу призводить до відмови ПЗ.

Описаний процес накопичення дефектів і помилок ПЗ, що призводить до погіршення характеристик продуктивності і надійності, називається процесом старіння та досліджується багатьма авторами із середини 90-х років XX-го століття: Парнас Д. Л., Тріведі К. С., Гротке М., Кінтала К., Котронео Д., Нателла Р., Хуан Й., Вайдянатан К., Шерешевський М., Сілва Л., Ахмад С., Руссо С., Федеріко Н., А. Авізеніс, Абдула З. Х., Яхая Дж. Х., Пєтрантуоно Р., Руссо С., Гарг С. та інші. Явище

старіння в реальних програмно-апаратних системах найчастіше проявляється як витоки пам'яті, не звільнені блокування файлів, пошкодження даних і чисельні накопичення помилок, що повільно погіршують продуктивність системи і, зрештою, спричиняють відмови у роботі ПЗ.

Визнаним економічно ефективним інструментом протидії явищу старіння є виконання так званої процедури омолодження ПЗ, яка полягає в очищенні стану системи від накопичених помилок. Наприклад, ручне чи автоматичне перезавантаження пристрою дозволяє повернути систему в стан із більшою продуктивністю та низькою імовірністю виникнення відмов старіння. Важливим науково-прикладним завданням є розроблення ефективних механізмів та стратегій виконання омолодження, а також, планування омолодження в оптимальний час, який дозволить максимізувати час перебування системи в стані з високою продуктивністю і надійністю, та мінімізувати ймовірність перебування системи в станах простою.

В сучасній науковій спільноті продовжує зростати інтерес до вивчення явища старіння ПЗ. Зокрема, шляхом дослідження особливостей процесу старіння, метрик старіння, факторів старіння в нових платформах, а також, розроблення методів омолодження, що враховують специфічні особливості конкретних систем. Мобільні пристрої є особливо вразливі до процесу старіння, оскільки користувачі часто використовують телефон тривалий час без перезавантаження, а їх технічні характеристики мають обмеження порівняно із ПК чи серверними системами. Мобільна ОС Android на даний момент досліджується в контексті старіння ПЗ багатьма вченими як на теоретичному, так і на емпіричному рівні, однак, це поле досліджень все ще знаходиться на ранньому етапі.

Таким чином, актуальною науково-практичною задачею є створення методів і засобів елімінації ефектів старіння в мобільних системах, зокрема в ОС Android, на основі емпіричного дослідження особливостей процесу старіння в цій ОС, що включає вивчення метрик та факторів старіння, створення більш достовірних та адекватних моделей процесу старіння, а також, розроблення методу омолодження ПЗ для ОС Android, який враховує як запропоновані теоретичні засоби, так і результати емпіричних досліджень.

Зв'язок роботи з науковими програмами, планами, темами. Дисертацію виконано на кафедрі програмного забезпечення Національного університету «Львівська політехніка». Тема дисертації відповідає науковому напрямку кафедри – програмне та математичне забезпечення автоматизованих систем. Зокрема, у 2021–2022 рр. дисертаційні дослідження виконувалися в межах держбюджетної науково-дослідної роботи «Розроблення інформаційної технології оцінювання та прогнозування надійності програмного забезпечення методами машинного навчання», номер держреєстрації 0121U109527.

Мета і завдання дослідження. Метою роботи є підвищення рівня надійності програмного забезпечення мобільних систем шляхом визначення факторів, що впливають на процес його старіння, побудови математичних моделей старіння і омолодження ПЗ та розроблення відповідних засобів.

Відповідно до вказаної у роботі мети **потрібно вирішити такі основні завдання**:

- Аналіз відомих підходів до оцінювання процесів старіння ПЗ та аналізу його впливу на надійність, встановлення їх переваг, недоліків та методів усунення ефектів старіння в програмно-апаратних системах;
- Розроблення моделей і методів оцінювання процесів старіння ПЗ для мобільних платформ реалізованих на основі native та cross-platform технологій;
- Встановлення факторів, що впливають на процеси старіння мобільних застосунків реалізованих з допомогою native та cross-platform технологій;
- Розроблення засобів усунення впливу відмов старіння на надійність ПЗ на різних етапах його життєвого циклу шляхом використання розроблених моделей і методів в процесі проектування, тестування та експлуатації ПЗ;
- Розроблення інформаційного забезпечення та програмних засобів на основі отриманих теоретичних результатів;
- Верифікація розробленого математичного та програмного забезпечення.

Об'єктом дослідження дисертаційної роботи є процес старіння та омолодження ПЗ.

Предметом дослідження є чинники та метрики старіння ПЗ, моделі старіння та омолодження ПЗ, методи омолодження ПЗ.

Методи дослідження. Для розв'язання сформульованих задач застосовано методи: теорії імовірностей та математичної статистики; методи обчислювальної математики; теорія марківських процесів; методи аналізу часових рядів; методи штучного інтелекту; методи системного аналізу; теорія алгоритмів та об'єктно орієнтована парадигма розроблення програмних засобів.

Наукова новизна отриманих результатів.

1. Вперше побудовано модель факторів старіння ПЗ для Android, яка відрізняється урахуванням метрик тривалості відображення кадру графічного інтерфейсу та частки пропущених кадрів графічного інтерфейсу, що дає змогу точніше виявляти ознаки процесу старіння ПЗ;

2. Вперше розроблено метод омолодження ПЗ для ОС Android на основі комплексної моделі старіння та омолодження, який відрізняється врахуванням активності користувача, різних рівнів старіння ПЗ та заряду батареї мобільного пристрою, що дає змогу визначити термін проведення омолодження ПЗ з найменшим впливом на функцію готовності та досвід користувача мобільного пристрою;

3. Отримали подальший розвиток моделі старіння та омолодження ПЗ з урахуванням активності використання мобільного пристрою користувачем та різних рівнів старіння, які відрізняються урахуванням переходу із стану старіння в стан омолодження та відсутністю переходу із стану очікування в стан активності під час "холодного" омолодження, урахуванням фактору заряду батареї, а також працездатних та непрацездатних станів, що дає змогу точніше оцінити вплив процесів старіння на показники надійності ПЗ та виконувати проектування щодо вибору параметрів методу омолодження ПЗ.

Практичне значення отриманих результатів.

1. Розроблено програмні засоби експериментальних досліджень процесу старіння в ОС Android: засоби виконання стресових тестів в ОС Android; засоби оброблення системних даних та формування часових рядів; засоби аналізу часових рядів метрик старіння.

2. Досліджено вплив факторів старіння: сценарії використання (із затримками генерування робочого навантаження та без затримок); типи застосунків (Android Native та Flutter cross-platform). Крос-платформові застосунки Flutter можуть бути більш вразливими до ефектів старіння в контексті продуктивності UI, ніж стандартні native застосунки. Сценарій використання із паузами характерний тим, що процес старіння може продовжуватись навіть без взаємодії користувача із UI, що говорить про необоротність процесів старіння в ОС без зовнішнього втручання та необхідність застосування омолодження саме в момент пауз активності для покращення досвіду користувача.

3. Досліджено старіння системних процесів та користувацьких застосунків ОС Android. Встановлено, що такі системні процеси як system_server, com.android.systemui та surfaceflinger можуть бути використані в методах омолодження ПЗ в якості індикаторів старіння. Такі процеси як cameraserver та audioserver можуть бути індикаторами в залежності від сценарію використання мобільного пристрою користувачем. Користувацькі застосунки також вразливі до явища старіння, однак виконання процедури омолодження може бути недоцільним в контексті надійності ПЗ, тому практичною рекомендацією є попередження виникнення дефектів старіння, а саме, попереджати помилки переповнення пам’яті та мінімізувати навантаження на основний потік UI.

4. Запропоновано варіант реалізації програмних засобів, що можуть бути використані на етапі проектування та розроблення користувацьких застосувань для протидії і реагування на процеси старіння в системі: функція для отримання даних про стан старіння системи та функція зворотного виклику у випадку виявлення системою процесів старіння. Розроблено структуру модулів програмного засобу омолодження ПЗ для ОС Android в контексті якої можливо реалізувати запропонований метод омолодження ПЗ, визначено основні вимоги до розроблюваного комплексу.

5. Запропонована в даній роботі модель старіння та омолодження може бути використана для оцінки таких показників якості існуючого ПЗ як надійність та продуктивність, для показників ефективності омолодження ПЗ, а також для

проектування та вибору параметрів методу омолодження ПЗ, що дає змогу формувати технічне завдання на його розроблення.

Особистий внесок здобувача полягає у формулюванні мети та основних завдань досліджень, обґрунтуванні наукових положень. Автором проаналізовано літературні джерела за темою дисертації, обґрунтовано напрями досліджень, розроблено моделі старіння та омолодження ПЗ і метод омолодження ПЗ для ОС Android, виконано експериментальні дослідження процесу, метрик та факторів старіння в ОС Android, систематизовано і узагальнено отримані результати. Робота містить прикладні положення та висновки, сформульовані дисертантом особисто. Ідеї, положення чи гіпотези інших авторів, які присутні в дисертації, мають відповідні посилання і використані лише для підкріплення ідей та результатів здобувача. Постановка завдань та їхнє обговорення здійснено під керівництвом д.т.н., проф. Яковини В.С.

Апробація матеріалів дисертації. Результати дисертаційного дослідження апробовано на міжнародних наукових та науково-практичних конференціях, наукових школах та консорціумах, семінарах:

1. IEEE 14th International Conference on Computer Sciences and Information Technologies. 2019.
2. IEEE 15th International Conference on Computer Sciences and Information Technologies. 2020.
3. XX Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах». 2020. 5-8 червня.
4. XXI Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах». 2021. 3-7 червня.
5. IEEE 16th International Conference on Computer Sciences and Information Technologies. 2021.

Публікації. За матеріалами дисертаційної роботи опубліковано 12 наукових праць, з яких 6 статей у наукових фахових виданнях України [1-6], 1 стаття в науковому періодичному виданні, яке включено до міжнародної наукометричної бази Web of Science [7], 5 – у матеріалах і тезах конференцій [8-12].

Структура й обсяг дисертації. Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел (118 найменувань) і 6 додатків. Основний зміст викладено на 136 сторінках друкованого тексту, містить 45 рисунків, 11 таблиць. Загальний обсяг роботи – 194 сторінки.

РОЗДІЛ 1. ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ ТА ПІДХОДІВ ДЛЯ ВИЯВЛЕННЯ ТА ЗМЕНШЕННЯ ВПЛИВУ ПРОЦЕСІВ СТАРІННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ЙОГО НАДІЙНІСТЬ

1.1. Поняття старіння програмного забезпечення та його характеристики

Поняття старіння ПЗ було впроваджено в середині 90-х років XX століття [13, 14]. Разом з тим, явище старіння ПЗ відоме досить давно, а ранні свідчення старіння ПЗ були знайдені вже в 1960-х роках: військова система Safeguard зазнала впливу зависань, що відбувалися після того, як були заповнені буфери повідомлень про помилки [15]. Прикладами дослідження явища старіння в реальних програмно-апаратних системах є дослідження комунікаційних та платіжних систем [13, 16], дослідження аварії протиракетної системи «Патріот», яка спричинила втрату людських життів [17], дослідження телекомунікаційних систем на початку 1990-х років [18]. Дослідження зосереджені як на теоретичному [19, 20], так і на емпіричному [21, 22, 23] розумінні процесу старіння ПЗ.

Явище старіння ПЗ – це процес накопичення помилок старіння та збільшення використання апаратних ресурсів в системах, які працюють тривалий час без перезавантаження, що призводить до погіршення продуктивності системи та збільшення відмов внаслідок старіння [13, 24, 25]. Старіння ПЗ має негативний вплив на такі характеристики якості [26] ПЗ як продуктивність та надійність [27, 28].

Для фізичних систем старіння відбувається у фазі зносу. Крива інтенсивності відмов [29] ілюструє таку поведінку збільшенням частоти відмов після певного стабільного періоду життя. Однак, в той час як апаратні несправності можуть виникнути через зношування, на перший погляд здається неможливим, що помилки в коді ПЗ, можуть спричиняти старіння ПЗ. Тому, для пояснення природи старіння ПЗ потрібно визначити основні поняття та концепції, зокрема з теорії надійності та систем.

Система – це сукупність взаємодіючих елементів, які забезпечують виконання визначених функцій. Межі системи відокремлюють її від середовища. Наприклад, єдина програмна система включає в себе АЗ, ОС і ПЗ як її елементи, однак

користувачі та інші системи є частиною її середовища. Кожна система може бути елементом іншої системи більш високого рівня.

Система може перебувати у двох основних станах, які характеризують її здатність виконувати свої функції, а саме, працездатний та непрацездатний стани. В працездатному стані система може виконувати усі функції у відповідності до її специфікації. В свою чергу непрацездатний стан може бути станом відмови або станом простою системи.

Відмова – це подія, що спричиняє порушення працездатного стану системи та унеможливлює виконання системою своїх функцій.

Стан простою системи полягає у тимчасовій відмові системи виконувати свої функції. В цьому стані можливе виконання робіт з обслуговування системи, зокрема, для виправлення та уникнення відмов системи.

Крім того, система може бути в стані погіршеної продуктивності, в якому виконання її функцій є повільним або обмеженим. В такому випадку мова йде про часткову відмову системи.

Природа явища старіння ПЗ пояснюється з точки зору так званого «ланцюга загроз» [24, 30] з урахуванням особливостей явища старіння ПЗ (рис. 1.1), що показує причинно-наслідкові зв'язки між дефектами, помилками і відмовами. Тобто, відмови, пов’язані із старінням, насправді є наслідками дефектів ПЗ.

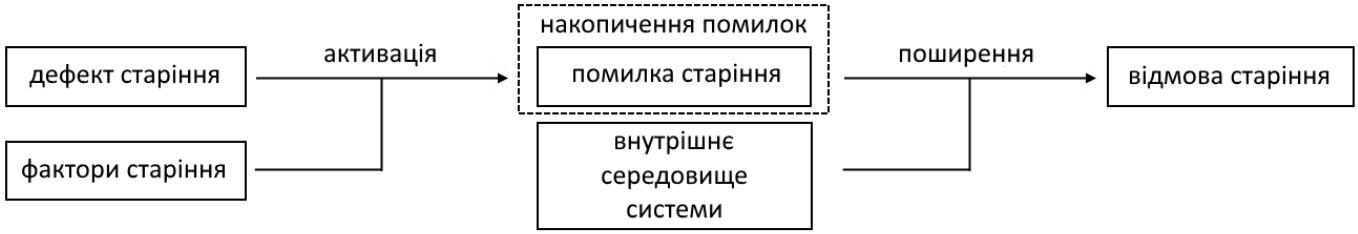


Рис. 1.1. Ланцюг загроз

Дефект старіння – це несправність у коді чи архітектурі ПЗ, що може спричинити помилку старіння. Дефект, який в даний час не призводить до помилки, вважається прихованим (латентним). Якщо певний набір вхідних даних є достатній

для активації прихованого дефекту, то це може спричинити виникнення помилки, що називається активацією дефекту. Інтенсивність, з якою активується прихований дефект, значною мірою залежить від активності та способу використання системи, а така кількісна характеристика називається операційним профілем [31].

В [30] використовується класифікація дефектів за їх здатністю до відтворення та розповсюдження. Несправності, чиї активація та поширення помилок є відтворюваними, називаються «жорсткими» (solid або hard), тоді як несправності, активація і поширення яких не є систематично відтворюваними, називаються «невловимими» або «м'якими» несправностями (elusive або soft).

Класифікація дефектів на Mandelbug і Bohrbug [32, 33] використовує більш об'єктивні критерії, пов'язані з властивостями дефектів. Дефекти групи Mandelbug важко ізолювати і можуть спричиняти відмови, які відтворюються не систематично. Наприклад, в коді програми відсутня ініціалізація змінної. Якщо програма-відлагоджувач може ініціалізувати ці змінні і запобігти виникненню несправності, то ця помилка є Mandelbug, оскільки програма-відлагоджувач є частиною внутрішнього середовища системи, що може впливати на активацію несправності. Bohrbug, з іншого боку, є легко ізольованою несправністю, яка завжди проявляється послідовно під чітко визначеним набором умов, а її активація і поширення не має «складності».

Всі помилки, пов'язані зі старінням, належать до групи помилок Mandelbug з двох причин. По-перше, може бути тривала затримка в часі між активацією несправностей і остаточним виникнення відмови. Саме така затримка дозволяє накопичувати помилки. По-друге, виникнення відмови, спричинене накопиченням помилки та поширенням помилок, може залежати від внутрішнього середовища системи.

Кожна помилка старіння викликана активацією дефекту. Помилка старіння - це частина внутрішнього стану системи, яка може призвести до виникнення відмови. Помилки можуть бути перетворені в інші помилки. Наприклад, помилка в компоненті 1 може досягати інтерфейсу обслуговування цього компонента, використовуваного компонентом 2, тим самим викликаючи помилку в цьому другому компоненті. Це перетворення помилок називається поширенням помилок. Поширення помилок

призводить до збою системи, якщо помилка поширюється на інтерфейс сервісу системи, спричиняючи відхилення наданого сервісу від специфікації. Термін поширення помилки використовується для перетворення помилки в іншу помилку як з причиною виникнення помилки, так і без неї.

Більшість помилок старіння, які ще не викликають відмову, зберігаються у внутрішньому стані системи. Як правило, саме це накопичення помилок, пов'язаних зі старінням, приводить внутрішнє середовище системи до стану, в якому розповсюдження помилок старіння викликають відмови старіння.

TTAF – це інтервал часу від моменту запуску системи або ініціалізації процесу до виникнення відмови старіння. Накопичення помилок зазвичай спричиняється поганим управління ресурсами, що призводить до їх вичерпання. Наприклад, витоки пам'яті, неперервані потоки та не звільнені блокування файлів. У таких випадках очікуваний час до відмови старіння називається часом до виснаження (TTRE). Крім того, на накопичення помилок впливає кількість і тип роботи, що виконується системою, яка називається робочим навантаженням.

Ефект старіння ПЗ полягає в поступовому переході від правильного внутрішнього стану до ймовірно помилкового або помилкового, тобто тоді, коли послідовна активація дефектів старінням викликає накопичення помилок старіння.

Хоча у багатьох випадках старіння ПЗ обумовлено помилками старіння, навіть за відсутності дефектів в коді ПЗ, ефекти старіння можуть виникати як наслідок природної поведінки системи. Цей вид старіння називається природним старінням. Прикладами природного старіння є проблеми фрагментації, які стосуються файлових систем, індексних файлів баз даних, а також основної фізичної пам'яті. Такі ефекти старіння не пов'язані з несправним кодом або дизайном архітектури, але вони є наслідком використання системи або застосунків.

Ефект старіння залежить від часу синхронізації (часу запуску системи або створення процесу), тільки якщо цей час є частиною внутрішнього середовища системи, що впливає на накопичення та поширення помилок [33].

Ефект старіння є необоротним без зовнішнього втручання. Наприклад, накопичені внутрішні помилки, спричинені послідовними активаціями пов'язаних зі

старінням дефектів, не зникають без зовнішнього втручання. В кращому випадку, ніякі подальші помилки не можуть накопичуватися в майбутньому протягом періодів в яких система не піддається впливу будь-яких чинників старіння [21].

Явище старіння проявляється в різних аспектах роботи ПЗ, тому було досліджено різні типи ефектів старіння та запропоновано [24, 27] класифікацію ефектів старіння, базуючись на їх загальних характеристиках. Зокрема, виділяють чотири основні класи ефектів, спричинених старінням на рівні ПЗ та ОС: витоки ресурсів (незвільнена пам’ять чи неперервані процеси та потоки), фрагментація (файлової системи та файлів бази даних), накопичення числових помилок (округлення), накопичення пошкоджень даних (файлова система та файли бази даних).

Ефекти старіння також можна класифікувати як стійкі та нестійкі [27]. Ефекти вважаються нестійкими, якщо їх можна видалити шляхом повторної ініціалізації системи або процесу, наприклад, через перезавантаження системи. В свою чергу, стійкі ефекти старіння продовжують існувати після повторної ініціалізації системи чи процесу. Фрагментація фізичної пам'яті та витік ресурсів ОС є прикладами нестійких ефектів старіння. Фрагментація файлової системи та метаданих бази даних є прикладами стійких ефектів старіння. Іншим прикладом стійкого ефекту старіння є накопичення числових помилок, що зберігається між перезавантаженнями системи через механізм контрольних точок. Потрібно зауважити, що так званий режим «сну» і подібні механізми очікування системи зберігають системну пам'ять, а, отже, ефекти старіння наявні в ньому навіть між повторною ініціалізацією системи.

У табл. 1.1 показано вплив старіння ПЗ в різних аспектах на основі даних літератури [27, 34]. Класифікація розширює попередні такими ефектами старіння ПЗ, що стосуються не тільки використання програмно-апаратних ресурсів, а й безпосередньо користувача і бізнесу.

Таблиця 1.1

Класифікація ефектів старіння ПЗ

Ефекти	**Пояснення**
Ефекти рівня ОС	Вплив старіння на ОС на рівні ресурсів: • Не звільнена пам’ять; • Помилки округлення і фрагментація даних файлів; • Помилки відлагодження.
Ефекти рівня користувацьких застосунків	Ефекти старіння на рівні застосунків: • Не звільнена пам’ять; • Незавершені потоки; • Помилки округлення; • Пошкодження файлів даних.
Досвід користувача	Вплив старіння на задоволення користувачів: • Згідно з дослідженням людино-машинної взаємодії, довгий час відгуку, який є більшим за 15 секунд, може негативно вплинути на задоволення користувачів; • Незадоволення користувачів призведе до зменшення та відмови від використання ПЗ.
Затримка роботи і графіка	Вплив старіння на роботу і графік: • Низька швидкість відгуку в запущених застосунках; • Час простою системи призводить до затримки роботи та відхилення від графіка.
Економічні ефекти	Економічні ефекти старіння: • Великі втрати грошей через старіння і відмови системи; • Багато грошей потрібно для придбання нового ПЗ.

Оскільки старіння ПЗ є серйозною проблемою, що впливає на його продуктивність та надійність, то важливими завданнями є протидія процесу старіння та мінімізація його ефектів. Можна виділити два основні підходи для протидії явищу старіння, а саме активний та пасивний підхід.

Омолодження ПЗ [13] – це активний підхід для боротьби із явищем старіння, який полягає у виконанні перезавантаження чи очищення процесів, системних компонент чи всього пристрою в запланований час, щоб зменшити кількість накопичених помилок старіння в стані системи і, тим самим, відтермінувати виникнення відмов старіння.

У свою чергу, пасивний підхід полягає у використанні інструментів на етапі проектування та розроблення ПЗ, застосуванні технік та засобів [34, 35] в самому ПЗ під час його експлуатації, що можуть зменшити імовірність виникнення помилок старіння чи дефектів старіння в коді ПЗ.

1.2. Фактори, що впливають на процес старіння та метрики для визначення наявності процесів старіння

Фактори старіння [24] – це чинники або їх комбінації, які впливають на активацію помилок старіння (рис. 1.1), що в свою чергу має прямий чи опосередкований вплив на процес старіння. Старіння є неминучим процесом у більшості випадків, тому, розуміючи фактори, які впливають на старіння ПЗ, можна стримати його виникнення та зменшити вплив на систему.

Фактори старіння ПЗ можна розділити на два типи [24]: зовнішні та внутрішні. До внутрішніх факторів старіння належать події та умови, що виникають всередині системи. Наприклад, функція викликає спрацьовування виконання тих частин коду, де знаходиться дефект старіння. Зокрема, до внутрішніх факторів належать дефекти пов’язані з відлагодженням, витоками пам’яті, переповненням пам’яті, незвільненими блокуваннями файлів, накопиченнями помилок округлення. В свою чергу, зовнішні фактори старіння – це тригери, які виконуються елементами в середовищі системи, наприклад, користувачами системи. Зовнішніми факторами [36-40] є динамічне середовище, функціональні вимоги та їх еволюція, покращення та

зміни функцій, стабільність бізнесу, вартість розроблення та обслуговування ПЗ, людський фактор, тощо.

Крім того, можна виділити три основні категорії зовнішніх факторів старіння ПЗ та відповідні їм метрики:

- Функціональні – продуктивність ПЗ, корисність, служба підтримки, час відгуку, помилки ПЗ;
- Середовища – служба підтримки, бізнес вимоги, зміни в бізнес процесах, зміни середовища, зміни технологій АПЗ;
- Людські – навчання, служба підтримки, популярність і технології, зміни у вищому керівництві, досвід.

Ще одним фактором старіння ПЗ є його складність, що в свою чергу вимірюється метриками програмного коду. Метрики ПЗ – це числова міра складності програми, що залежить від властивостей тексту програми [41] і обчислюється статично без необхідності виконання програми. Розмір ПЗ, його складність, використання деяких видів структур програмування, пов'язаних з управлінням ресурсами, використання арифметичних операторів та інші особливості пов'язані з виникненням помилок. Можливість пов'язати складність ПЗ зі старінням ПЗ може бути корисною для зменшення проблеми старіння ПЗ. Наприклад, оцінюючи ступінь старіння в коді ПЗ, розробники могли б виконати необхідні тести для виявлення та виправлення помилок старіння в процесі розробки. Більш того, оцінка старіння на основі метрик ПЗ може бути корисною для уточнення аналітичних моделей для вибору найкращого графіка омолодження ПЗ.

У роботі [42] досліджується, чи залежить старіння ПЗ від метрик ПЗ. Для того, щоб з'ясувати, чи є в програмі якісь характеристики ПЗ, які можуть викликати або просто вказувати на помилки, пов'язані зі старінням, створено і виконано емпіричне дослідження, яке базується на десяти застосунках, чиї проблеми старіння ПЗ відомі. Результати цього дослідження показують і підтверджують, що ефекти старіння ПЗ пов'язані зі статичними характеристиками ПЗ.

Ефекти старіння в системі можна виявити тільки під час роботи системи за допомогою метрик та індикаторів старіння, тобто системних змінних, які можуть бути

безпосередньо виміряні і пов’язані з явищами старіння ПЗ. Прикладами метрик старіння є дані про використання ресурсів ОС, такі як вільна фізична пам'ять, використовуваний простір підкачки, розмір таблиць файлів і процесів [13]. Метрики старіння – це змінні, які індивідуально або в комбінації можуть свідчити про те, чи спостерігається процес старіння в стані системи. Вони можуть розглядатися на декількох рівнях, таких як ОС та її компоненти, процеси користувацьких застосунків та проміжне ПЗ (middleware), віртуальна машина (Virtual Machine) і монітор віртуальної машини (Virtual Machine Monitor).

Виділяють два загальні класи показників старіння [24], відповідно до їх деталізації: загальносистемні метрики та метрики застосунків.

Загальносистемні метрики надають інформацію, що стосується підсистем, які використовуються кількома запущеними програмами. Прикладами загальних підсистем є ОС, проміжне ПЗ, віртуальні машини і менеджер віртуальних машин та інші. Метрики цієї категорії часто використовуються для оцінки ефектів старіння на систему в цілому, а не для конкретного застосування. Прикладами показників старіння в цій категорії є вільна фізична пам'ять, використовуваний простір підкачки, розмір файлу таблиці та навантаження на CPU.

Індикатори застосунків надають конкретну інформацію про окремий процес застосування, тим самим даючи більш точну інформацію про нього, ніж загальносистемні індикатори. Коли процес застосунку виконується під віртуальною машиною (наприклад, програми Java), то показники старіння віртуальної машини також можуть використовуватися для оцінки програми, що виконується у віртуальній машині. Прикладами метрик старіння в цій категорії є RSS, розмір купи Java VM і час відгуку інтерфейсу користувача.

1.3. Методи дослідження та моделі старіння ПЗ.

Згідно [43], всі методи дослідження старіння ПЗ можна розділити на дослідження на основі моделей, дослідження на основі вимірювань, гібридні дослідження, дослідження стратегій планування омолодження ПЗ, дослідження методів омолодження і дослідження відмов старіння в промислових даних. Ще однією класифікацією досліджень [44] є їх поділ на аналітичні, емпіричні та гібридні.

Методи дослідження на основі моделей виконують моделювання явища старіння ПЗ для того, щоб забезпечити абстрактне уявлення про нього і зробити його застосовним для математичної обробки. Використання та розробка моделей відіграє важливу роль у процесі дослідження явища старіння та омолодження ПЗ. Моделі використовуються для різних цілей, зокрема, таких як: визначення характеристик старіння ПЗ, моделювання та прогнозування процесу, а також для оцінки планування часу виконання омолодження та витрат на його виконання. Аналітичні моделі вперше були використані для того, щоб довести, що омолодження ПЗ може зменшити вартість простою системи [13] і мінімізувати час завершення програми [45] при наявності явища старіння.

Дослідження на основі моделей можна поділяти [43] за типом стохастичного процесу, який використовується для моделювання явища. Існує багато типових моделей, як правило, на основі ланцюгів Маркова, до яких відносяться Марківські, напівмарківські та інші Марківські процеси, а також, мережі Петрі.

Марківські і напівмарківські процеси є основою для багатьох моделей. Перша робота [13] про старіння ПЗ моделювала явище за допомогою СТМС, який є базовою моделлю для опису явища. Ця базова модель розширена багатьма способами, наприклад, в [46] використовується неоднорідний СТМС, де час перебування в кожному стані є неекспоненціально розподіленим, а в [19] використовуються напівмарківські процеси, де інтенсивності переходу з одного стану в інший залежать, окрім поточного стану, від часу перебування в цьому стані.

Марківські ланцюги широко використовуються у своїй основній формі для:

1) аналізу складних систем з декількома стратегіями омолодження (наприклад, в [47] поведінка кластерних систем описана з допомогою СТМС);

2) аналізу старіння в нових контекстах, таких як системи, що використовують технології віртуалізації (приклади наведені в роботах [48, 49], де старіння і омолодження вивчаються для віртуалізованих серверів);

3) опису більш складних проявів відмов (замість моделювання одного стану відмови, що призводить до непрацездатного стану, розглядаються різні ступені відмов для моделювання поступового зниження продуктивності [50-53]).

Хоча більшість досліджень застосовують класичні Марківські і напівмарковські процеси, є і інші типи моделювання. Наприклад, автори [46, 54, 55] використовували MRGP у поєднанні зі SPN, щоб побудувати просту, але загальну модель для оцінки оптимального графіка омолодження в програмній системі. Марковські регенераційні процеси є узагальненням Марківських і напівмарковських процесів, які можуть фіксувати поведінку реальних систем з детермінованими і експоненціально розподіленими часом подій.

Проблема знаходження оптимального графіка омолодження була сформульована також як MDP, де: час дискретизується і представляє один вимір опису стану, рішення в кожному стані визначають, чи повинна система бути омолодженою, чи ні, і вирішення полягає в пошуку оптимальної стратегії, щоб мінімізувати функцію витрат. Приклад представлений в [51], де автори адаптували процес прийняття рішень Маркова для побудови моделі омолодження ПЗ в телекомунікаційній системі, що включає виникнення переповнення буфера. В [53] POMDP також використовуються для моделювання явища.

Мережі Петрі і їх численні варіанти – це формалізми, строго пов'язані з Марківськими моделями, але є більш компактними і в деяких випадках легшими для визначення. Цей формалізм дозволяє більш легко виразити показники продуктивності з кількома рівнями і є особливо корисним для вираження метрик у більш складних системах, таких як системи з багатьма вузлами (наприклад, кластери).

Наприклад, DSPN використовувалися в [55] з метою побудови моделі для аналізу продуктивності кластерних систем при різному навантаженні. Аналогічно, SRN використовувалися в [56] для моделювання кластерних систем. В [57] був представлений підхід, заснований на мережах Петрі, в якому автори зосереджувалися на оцінці стратегії омолодження системи, що спрацьовує за часом, з використанням моделі черги, сформульованої як розширена SPN. В роботі [58] поєднано SRN з SysML [59], щоб спростити системним адміністраторам аналіз старіння та омолодження у своїх серверних системах. Метою авторів [58] є застосування напівформальної мови SysML для опису конфігурацій системи та операцій технічного обслуговування, що дозволяє людям, які не мають досвіду моделювання надійності,

розробляти та вивчати вплив різних стратегій омолодження, розгорнутих у серверних системах.

Дослідження на основі моделей аналізують абстрактні моделі, роблячи деякі спрощені припущення про систему, такі як припущення про базові стохастичні розподіли, що характеризують систему. Ці моделі можуть застосовуватися до широкого кола систем, тому підходи на основі моделей можуть надавати більш загальні висновки і можуть бути застосовними для всіх систем, ніж підходи на основі вимірювань. Тим не менш, омолодження ПЗ на основі моделей може бути менш ефективним, ніж на основі вимірювань, оскільки воно може не використовувати деякі особливості конкретної системи, і може бути не в змозі адаптуватися до неочікуваних умов. У будь-якому випадку, підходи, що базуються на моделях, покладаються на реальні дані для того, щоб заповнити параметри моделі, які можна отримати з аналізу на основі вимірювань.

В методах дослідження на основі вимірювань значна увага приділяється емпіричному аналізу старіння ПЗ, що ґрунтується на вимірюванні метрик реальних систем. Оскільки явище старіння проявляється як погіршення продуктивності та споживання ресурсів [29, 60], то дослідження зосереджуються на підходах емпіричних вимірювань систем для того, щоб визначити, чи знаходиться система в стані схильному до відмов старіння ПЗ, для прогнозування часу до відмови старіння і планування омолодження ПЗ. Дослідження на основі вимірювань також надають детальну інформацію про явище старіння в реальних системах, що корисно для кращого розуміння природи та масштабів старіння ПЗ.

Підхід омолодження ПЗ на основі вимірювань полягає в безпосередньому моніторингу системних індикаторів, які можуть вказувати на початок старіння ПЗ, і прогнозуванні виникнення відмов старіння шляхом статистичного аналізу зібраних даних під час виконання. Моніторинг та прогнозування забезпечують даними для визначення найкращого часу проведення омолодження. Можна виділити наступні групи підходів для прогнозування: аналіз часових рядів; машинне навчання; підходи на основі порогів.

Аналіз часових рядів широко застосовується під час моніторингу ресурсів. Часові ряди аналізуються, як правило, використовуючи тести трендів, щоб прийняти чи відхилити гіпотезу про наявність тренду в даних (наприклад, Mann-Kendall, t-student, Seasonal Kendall tests), і підходи для оцінки таких трендів та можливої сезонності в даних (наприклад, множинна лінійна регресія, згладжування регресії, процедура оцінки нахилу Сена, авторегресивні моделі).

В [61] протягом 9 днів здійснювався моніторинг набору дев’яти робочих станцій Unix за допомогою інструменту моніторингу на основі SNMP. Протягом періоду спостереження 33% повідомлень про відмови були пов'язані з виснаженням ресурсів, що підкреслює те, що старіння ПЗ є значним джерелом збоїв у комп'ютерних системах. В [21, 62] проаналізовано зниження продуктивності на веб-сервері Apache шляхом вибірки часу відповіді веб-сервера на попередньо визначені HTTP-запити з фіксованими інтервалами. В [21] також розглядався аналіз сезонних шаблонів, в якому тренди аналізуються також при наявності сезонних коливань даних. Моделі часових рядів ARMA та ARX використовувалися в [62] на веб-сервері Apache, щоб оцінити вичерпання ресурсів. У порівнянні з лінійною регресією і розширеними лінійними регресійними моделями, ARX модель тягне за собою більш високі початкові накладні витрати, але як тільки вона буде побудована, то може бути використана для прогнозування протягом тривалого періоду без повторної перевірки параметрів моделі.

Аналіз часових рядів також застосовується для вивчення взаємозв'язку явища старіння з робочим навантаженням у складних системах, включно з кодом ядра Linux [63] і віртуальною машиною Java [64]. В обох випадках аналіз параметрів робочого навантаження використовується для надання свідчень про потенційні джерела старіння ПЗ, виділяючи підсистеми, параметри яких корелюють з тенденціями старіння. PCA з подальшою множинною лінійною регресією застосовується з метою усунення кореляції першого порядку між показниками, а потім для забезпечення лінійних оцінок трендів старіння, з допомогою регресії, зменшуючи проблеми мультиколінеарності.

Моделі ARIMA та Holt-Winters (потрійне експоненціальне згладжування) були використані в [65], де було розроблено фреймворк для виявлення аномалій продуктивності, викликаних старінням, які спрямовані на веб- та компонентні застосунки. Зокрема, фреймворк відстежує параметри програми чи системи, які використовуються для визначення кореляції між часом відгуку програми та вхідним навантаженням, що в свою чергу використовується для тренування алгоритмів машинного навчання. Під час виконання системи, параметри, зібрані за допомогою моніторингу, оцінюються алгоритми ARMA і Holt-Winters, а потім ці оцінки, класифіковані за допомогою тренованих алгоритмів машинного навчання, щоб визначити, чи може застосунок зазнати певної аномалії продуктивності.

У [66] використовувалися чотири різні моделі часових рядів для належного планування оновлення ПЗ: лінійна модель, квадратична модель, модель експоненційного зростання і модель логістики Pearl-Reed. Вони були застосовані для прогнозування трендів використання пам'яті в системі хмарних обчислень Eucalyptus [67].

В [68] багатоваріантні нелінійні моделі (метод опорних векторів, радіальні та універсальні базисні функції) були порівняні з багатовимірними лінійними моделями. Перші показали кращу продуктивність, ніж лінійні моделі під час порівняльного аналізу в тематичних дослідженнях.

Підходи машинного навчання є більш досконалою формою аналізу даних, які застосовують алгоритми з області штучного інтелекту (наприклад, класифікатори і регресори) для виявлення трендів і класифікації стану системи як надійної або схильної до відмов. В [69] автори застосовують методи розпізнавання образів для прогнозування явищ старіння ПЗ в загальному пулі пам'яті великих OLTP-серверів. Підхід застосовує нелінійну непараметричну регресію до великого набору системних змінних і аналізує залишкову помилку між прогнозованими та фактичними значеннями системи за допомогою послідовного тестування коефіцієнта ймовірності, щоб передбачити початок проявів старіння. Результати показали, що ці методи дозволили виявити значні відхилення від «стандартної» поведінки з 2-годинним раннім попередженням. Інший приклад застосування підходу машинного навчання

для прогнозування відмов старіння ПЗ досліджено в [70] у контексті трирівневої системи J2EE, де запропоновано підхід машинного навчання для побудови автоматичних моделей регресійних дерев, які пов’язують декілька системних змінних (наприклад, кількість з'єднань і пропускну здатність) з трендами старіння, на основі спостереження, що тенденції старіння ПЗ можна апроксимувати за допомогою кусково-лінійної моделі. Моделі навчалися з використанням зразків даних, зібраних у попередніх експериментах, і використовувалися для прогнозування TTRE в умовах, відмінних від тих, що спостерігалися під час фази навчання. Три різні алгоритми машинного навчання (наївний класифікатор Байеса, дерева рішень і модель нейронної мережі) також використовувалися у поєднанні з моделями часових рядів [65], щоб передбачити старіння у веб-застосунках.

Підходи на основі порогів визначають пороги для деяких показників старіння, і омолодження викликається тоді, коли контрольовані показники перевищують такі пороги. Наприклад, показники можуть стосуватися використання ресурсів. Труднощі виникають у визначені найкращих показників і правильних порогів для них, які здатні запобігти одночасному виникненню відмов і виконанню процесу омолодження ПЗ. Прикладом такого підходу є робота [71], яка застосовує пороги щодо середнього часу відгуку та показників якості послуг. В статті пропонується підхід до омолодження, заснований на методах самовідновлення, який використовує віртуалізацію для оптимізації відновлення. Вони реалізували так званий фреймворк омолодження VM-Rejuv, де модуль виявлення старіння знаходить умови старіння на основі згаданих порогових значень.

Дослідження на основі вимірювань прогнозують старіння ПЗ на основі прямих вимірювань (наприклад, на основі часових рядів) і надають емпіричні дані про явища старіння ПЗ. Перевагою такого підходу є те, що прогнозування старіння ПЗ може адаптуватися до поточного стану системи (наприклад, поточний робочий профіль, який, можливо, не був передбачений до початку експлуатації), і може точно передбачити виникнення явища старіння. Однак, такий підхід не може бути легко узагальненим, оскільки він використовує певний особливий аспект, пов’язаний з природою розглянутої системи. Наприклад, той факт, що певний ресурс має сезонні

або фрактальні закономірності [20, 61]. Крім того, підходи на основі вимірювань не призначені для оцінки довгострокових характеристик надійності, таких як доступність.

Гібридні методи та дослідження поєднують переваги підходів на основі моделей і вимірювань, а саме, описують явище аналітично, найчастіше за допомогою марковських моделей, і визначають параметри моделі за допомогою вимірювань системних показників. Незважаючи на практичну важливість гібридних підходів, було зроблено лише невелику кількість таких досліджень [43].

В роботах [72] та [73] представлено результати аналізу, проведеного на тому ж самому наборі робочих станцій Unix, що використовуються в [61]. В той час, як в останній роботі розглядалися лише виявлення та оцінка вичерпання ресурсів за часом, не враховуючи навантаження, перші дві роботи враховували навантаження на систему та будували модель для оцінки TTRE. Вони розглядали деякі параметри для включення робочого навантаження в аналіз, наприклад, кількість перемикань контексту процесора і кількість системних викликів. Спочатку були визначені різні стани робочого навантаження через статистичний кластерний аналіз і побудована модель простору станів, що визначає розподіл часу перебування. Далі для моделі була визначена функція винагороди, заснована на швидкості вичерпання ресурсів для кожного стану робочого навантаження. Розв’язуючи модель, отримано тренди виснаження ресурсів і TTRE для кожного розглянутого ресурсу в кожному стані робочого навантаження. Методологія дозволяє проводити характеристику старіння, обумовлену навантаженням. Друга з цих двох робіт [73] є більш чітким прикладом гібридного підходу, де:

1) побудовано напівмарковську модель, засновану на вимірюваннях;

2) обчислено час до виснаження для кожного розглянутого ресурсу та стану;

3) пропонується напівмарківська модель доступності, яка базується на реальних даних, а не на припущеннях про поведінку системи.

У всіх трьох класах досліджень ще одним фактором, який слід враховувати, є включення в аналіз впливу робочого навантаження. Старіння чітко співвідноситься з варіюванням робочого навантаження. Багато авторів [19, 73, 74, 75] розглядали

робоче навантаження в своїх дослідженнях і також змінювали оцінку TTAF в заданий час як функцію навантаження, яке фактично зазнає система. Для цієї мети було враховано декілька різних підходів, наприклад, у обговореному дослідженні [73] змодельовано навантаження у вигляді напівмарковського процесу, тоді як у [22, 75, 76] розроблену експериментальну техніку використано для планування експериментів з різним навантаженням тільки на основі вимірювань.

Дослідження методів уникнення помилок старіння шляхом верифікації та відлагодження ПЗ є ще одним напрямком в інженерії ПЗ. Ці дослідження вирішують такі задачі як перевірка, тестування, відлагодження та уникнення дефектів, що можуть активізувати помилки старіння. Прикладами є роботи з статичного та динамічного аналізу програмного коду для виявлення дефектів старіння, таких як витоки пам'яті [77, 78, 79, 80].

Деякі дослідження аналізують помилки та відмови старіння з реальних джерел даних, а не з контрольованих експериментів. Наприклад, в [81] проаналізовано звіти з космічних місій NASA та JPL і їх помилки класифіковано як Bohrbugs, Mandelbugs та помилки старіння. У цьому випадку проводиться аналіз джерела старіння, тобто помилок старіння.

Підходи досліджень на основі вимірювань можуть бути більш дорогими для виконання, однак вони забезпечують об’єктивними даними старіння ПЗ в конкретній системі. Обидва методи досліджень старіння, а саме, на основі моделей і вимірювань, необхідні для вивчення явища старіння ПЗ в широкому спектрі. Однак, за результатами досліджень літератури в [26] та [27] дослідники роблять висновок, що гібридні підходи представляють важливу роль і заслуговують на більшу увагу, оскільки вони поєднують переваги підходів на основі моделей та на основі вимірювань для отримання корисних даних про процес старіння, ефективних стратегій виконання та планування омолодження ПЗ.

1.4. Методи виконання процедури омолодження ПЗ

Витоки ресурсів та інші ефекти старіння ПЗ можуть бути пов'язані з дефектами та помилками старіння в прикладному ПЗ, в бібліотеках, які використовуються у прикладному ПЗ, або в середовищі виконання застосунків, наприклад, ОС. Однак,

виправлення цих дефектів не завжди є можливим, оскільки помилки старіння можуть бути в сторонніх або повторно використовуваних програмних кодах, які є не доступні для розробників. Більше того, виявити помилки старіння в складних програмних системах може бути дуже важко. Для вирішення описаних проблем в контексті старіння ПЗ активно досліджується та використовується так зване омолодження ПЗ.

Омолодження ПЗ [13, 82] – це активна техніка запобігання і затримки старіння ПЗ. Цей підхід передбачає регулярні чи нерегулярні скидання внутрішнього стану системи таким чином, щоб очистити накопичені помилки старіння ПЗ. Наприклад, шляхом перезапуску ОС відбувається повернення в «чистий» початковий стан. Омолодження ПЗ представляє собою форму супроводу ПЗ, що відрізняється від таких підходів як установка оновлень для виправлення помилок [83] або реінжиніринг ПЗ для вирішення проблем морального старіння [14].

Простим прикладом омолодження ПЗ є перезапуск програми чи ОС, що забезпечує очищення структур даних в пам’яті, повернення процесів у початковий стан або в стан попередньої контрольної точки. Омолодження ПЗ може бути реалізовано на різних рівнях деталізації та застосовано до багатьох типів елементів у системі, таких як ОС, окремі процеси системних сервісів чи прикладного ПЗ, або постійні об'єкти даних, такі як метадані файлової системи та файли індексів баз даних. При цьому, системні процеси, сервіси та користувацькі застосунки можуть розглядатися як об’єкти спостереження для виявлення старіння та планування процедури омолодження, так і як цільові об’єкти для виконання перезавантаження чи очищення.

Під час досліджень ПЗ в лабораторіях AT&T Bell [60, 13, 84, 15, 43] омолодження ПЗ визначили як економічно ефективне рішення для усунення ефектів старіння і уникнення відмов, пов'язаних зі старінням, які не вимагають знання про місця помилок, пов'язаних зі старінням, або навіть самого факту їх існування.

Оскільки програма може бути недоступною під час виконання процедури омолодження, то це може збільшити час простою і спричинити певні витрати. Однак, ці витрати можуть бути зведені до мінімуму шляхом планування омолодження, щоб забезпечити найменший час простою. Таким чином, витрати простою будуть

високими, якщо час простою не запланований, як це відбувається у випадку виникнення відмови. Омолодження може уникнути або принаймні відкласти виникнення відмов, пов'язаних зі старінням, тому воно може зменшити загальний час простою та пов'язані з цим витрати.

З описаної вище причини випливає, що найважливішою проблемою омолодження ПЗ є планування його виконання під час роботи ОС, щоб покращити характеристики надійності і зменшити витрати часу і ресурсів системи. В [13] представлено просту загальну модель старіння та омолодження на основі СТМС для аналізу омолодження ПЗ (рис. 1.2). У цій моделі після запуску система залишається в так званому «дуже надійному стані» S_0 в якому ймовірність відмов старіння є незначною. Через деякий час із інтенсивністю a_{0P} система перейде в стан S_P, який характеризується високою імовірністю відмов старіння. У такому стані відмова старіння може відбутись з інтенсивністю a_{PF} і відбудеться перехід в стан відмови S_F з наступним відновленням до стану S_0 з інтенсивністю a_{F0}. Якщо система виконує омолодження, вона перейде зі стану S_P в S_R з інтенсивністю a_{PR}, а потім до надійного стану S_0 з інтенсивністю a_{R0}. Ця модель дозволяє обчислити очікуваний час простою та їх витрати, що, в свою чергу, дозволяє проаналізувати в яких умовах омолодження ПЗ є вигідним для забезпечення надійності. Наприклад, якщо вартість омолодження невелика, а частота відмов велика, омолодження слід виконувати тоді, коли застосунок перейде у стан S_P. Крім того, в дослідженнях старіння та омолодження ПЗ запропоновано інші моделі, які були розглянуті в підрозділі 1.3.

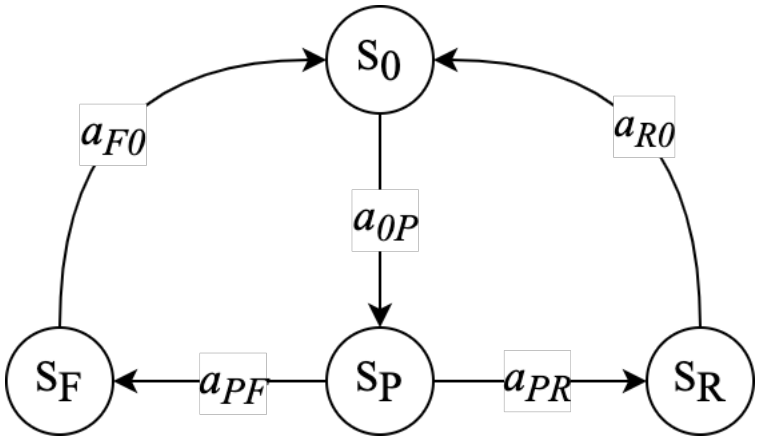


Рис. 1.2. Граф станів і переходів базової моделі старіння та омолодження ПЗ

Приклади систем, які використовують методи омолодження ПЗ: системи обробки транзакцій, веб-сервери, системи космічних апаратів.

Реальним прикладом може бути метод омолодження ПЗ веб-сервера Apache, який реалізує одну з форм омолодження, що завершує і перестворює процеси після виконання певної кількості запитів [85]. Ще один підхід полягає в перезавантаженні віртуальних машин, які працюють у середовищі хмарних обчислень [86]. Телекомунікаційна корпорація AT&T впровадила омолодження ПЗ в режимі реального часу, збираючи дані виставлення рахунків у США для більшості телефонних станцій [87].

Актуальні дослідження старіння та омолодження ПЗ [88, 89] розглядають виконання процедури омолодження на трьох рівнях, а саме, на рівні АЗ, ОС та прикладного ПЗ. Альтернативною класифікацією є поділ механізмів на «тепле» та «холодне» омолодження [88]. «Холодне» омолодження на рівні АЗ відбувається шляхом перезавантаження всього пристрою, що дає найкращий результат покращення продуктивності системи, але займає найбільше часу і може перервати роботу користувача. В свою чергу, омолодження шляхом перезавантаження прикладного ПЗ («тепле» омолодження) займає найменше часу, однак результативність покращення продуктивності невисока і, також, можливе втручання в діяльність користувача чи у виконання важливих задач. Одним із компромісних рішень є виконання процедури омолодження на рівні ОС шляхом перезавантаження основних системних сервісів відповідальних за старіння, що також є варіантом «теплого» омолодження. Отже, техніки «теплого» омолодження на відміну від «холодного» омолодження можуть бути менш ефективними, але дозволяють зберегти систему в працездатному стані без виконання перезавантаження або зменшити час простою системи під час виконання процедури омолодження.

1.5. Аналіз старіння ПЗ в мобільних ОС.

Старіння ПЗ проявляється не тільки в системному або прикладному ПЗ серверів чи ПК, але і в мобільних застосуваннях таких ОС як Android [90, 91]. Мобільні пристрої безперервно використовуються користувачами протягом тривалого часу без перезавантаження, а їх апаратні ресурси обмежені в порівнянні із ПК та серверами.

Таким чином, мобільні пристрої є особливо вразливими до впливів старіння ПЗ [1, 2]. Крім того, мобільні пристрої мають особливості, які відрізняють їх від інших програмно-апаратних систем і мають бути враховані в дослідженнях явища старіння ПЗ, наприклад, залежність роботи від заряду батареї.

Відкрита мобільна платформа Android на базі ядра ОС Linux є найпопулярнішою мобільною системою і станом на 2022 рік займає 85,9% [91] ринку мобільних систем. Попередні дослідження явища старіння в основному зосереджені на ОС Linux, а дослідження ОС Android знаходяться на ранній стадії. Хоча ОС Android є Linux-системою, неможливо повністю повторювати схеми досліджень Linux. Наприклад, простір підкачки є важливим показником у виявленні старіння ПЗ Linux, але він не підходить для Android [93].

Для Android платформи було розроблено різні методології та підходи для дослідження метрик, факторів та процесів старіння ПЗ. В [94] запропоновано підхід дослідження старіння ПЗ системи Android, який використовує існуючі інструменти для моніторингу та виявлення витоків пам’яті в застосунках Android. Інструмент Monkey [95] використовувався для створення стресового навантаження, метою якого є прискорення виникнення старіння ПЗ. Утиліти Linux використовувалися для збору інформації про використання ресурсів пристрою. Зв’язок між ПК і мобільним пристроєм забезпечувався за допомогою інструменту ADB [96]. Результати [94] вказують на ефективність такого підходу і дозволяють виявляти ефекти старіння ПЗ, такі як витоки пам'яті.

У роботах [89, 91] представлено розгорнуту і допрацьовану експериментальну методологію для аналізу старіння ПЗ в ОС Android. Методологія використовує статистичні методи, щоб визначити, які фактори (такі як робочі навантаження та конфігурації пристроїв) посилюють погіршення продуктивності та споживання ресурсів. Крім того, методологія аналізує взаємозв'язок між старінням ПЗ та показниками використання ресурсів, щоб визначити, які підсистеми впливають на старіння та забезпечити розробку стратегій омолодження ПЗ. Для визначення наявності трендів старіння в роботі використовуються метод Манна-Кандела та процедура Сена для визначення нахилу трендів.

Старіння ПЗ дійсно впливає на пристрої Android, що було підтверджено в експериментальних дослідженнях. Зокрема, після декількох годин стрес-тестування пристрої відчули помітне погіршення продуктивності з точки зору збільшення часу відгуку системи. В роботі визначено, що на продуктивність впливають такі фактори як робоче навантаження (програми, події) і конфігурація (зокрема, доступний об’єм пам’яті для зберігання даних).

Проблеми старіння ПЗ можуть бути пов'язані з конкретними процесами в ОС Android, включаючи System Server, System UI і Surface Flinger, які демонструють завищені витрати пам’яті. Крім того, специфічні сервіси всередині них, такі як Activity Manager та Power Manager, демонструють тренди старіння.

Тренди погіршення продуктивності корелюють з показниками використання ресурсів ядра, такі як розмір PSS процесів і сторінок, об’єднаних з допомогою KSM, і з часом, витраченим на збір сміття. Ці показники можуть бути використані як метрики старіння ПЗ, а також для планування омолодження ПЗ в ОС Android.

Доменіко Котронео і співавтори в 2016 році [91] ідентифікували дві групи метрик, що дозволяють узагальнити метрики для вивчення старіння в цілому, так і для ОС Android: метрики, що сприймаються користувачем та системні метрики.

Метрики, що сприймаються користувачем, також можна назвати метриками користувацького інтерфейсу, оскільки вони дозволяють оцінити відгук системи на події вводу користувача та продуктивність інтерфейсу в цілому.

Системні метрики – це індикатори використання таких системних ресурсів як оперативна пам’ять та сховище даних, а також такі системні операції, як збирач сміття та менеджер процесів. Порівняння і аналіз системних метрик разом із метриками інтерфейсу дозволяє ідентифікувати ділянки ОС Android, які є такими, що піддаються ефектам старіння ПЗ.

Деякі алгоритми машинного навчання використовувалися для вивчення характеристик старіння ПЗ, такі як дерево прийняття рішень [65, 97, 98, 99, 100], SVM [100], нейронні мережі [65], і так далі. У роботі [101] було розроблено експерименти для порівняння ефективності трьох алгоритмів машинного навчання (ID3, SVM, DBN) і впроваджено новий показник (page fault) для виявлення старіння ПЗ в системі

Android. Було порівняно ефективність трьох алгоритмів машинного навчання при виявленні старіння ПЗ. Експериментальні результати показали, що дерево рішень і SVM є більш ефективними, ніж DBN. Нововведений індикатор помилки сторінки (page fault) в експериментах показав, що стабільність моделей з використанням цього індикатора старіння є трохи кращою за ті моделі, що використовують час запуску для виявлення старіння ПЗ в системі Android, що означає, що він також може бути використаний як хороший індикатор старіння Android.

Крім дослідження процесів, метрик та факторів старіння, для ОС Android також пропонуються різні підходи зменшення впливу старіння та використання технік омолодження ПЗ. Наприклад, в роботі [102] запропоновано і реалізовано засіб виявлення старіння та виконання процедури омолодження для ОС Android (ADARTA), який виконує моніторинг системних процесів та трендів збільшення навантаження на систему, визначає стан старіння системи, оцінює TTAF, планує та виконує омолодження ПЗ шляхом перезавантаження основних системних сервісів.

Крім виконання процедури омолодження ПЗ досліджуються механізми протидії старінню в межах прикладних застосунків шляхом зменшення робочого навантаження та уникнення помилок старіння. Наприклад, алгоритм розвантаження обчислень [35] має на меті розділити програму на локальні (обчислення виконуються на мобільному пристрої) та віддалені (виконання складних обчислень відбувається на віддаленому сервері) частини для того, щоб зменшити загальну тривалість виконання програми локально і тим самим уникнути надлишкового навантаження, накопичення помилок старіння та споживання заряду батареї. Таке розвантаження є корисним, якщо віддалене виконання має кращу продуктивність, ніж виконання на пристрої, або, що рівнозначно, якщо витрати на передачу даних на віддалений сервер менші, ніж виграш у часі або використання енергії під час локального виконання обчислень. Стверджується, що схема часткового розвантаження здатна ефективно зменшити час виконання програми, а також споживання енергії. Крім того, вона може певною мірою адаптуватися до змін у середовищі та уникнути різкого зниження продуктивності застосунків після різкого падіння пропускної здатності.

Крім емпіричних досліджень явища старіння та розроблення методів омолодження, для ОС Android виконуються теоретичні дослідження та розробляються аналітичні моделі, які описують процес старіння та омолодження з урахуванням особливостей мобільних пристроїв та систем. Наприклад, для ОС Android побудовано модель старіння та омолодження у вигляді СТМС [103], яка враховує активність використання телефону користувачем для того, щоб можна було передбачити поведінку користувача у різних часових інтервалах дня. Використовуючи SPN [104] і інструмент Oris [105], було згенеровано СТМС, яка об'єднує модель активності користувача та модель старіння із виконанням омолодження ПЗ. Завдяки отриманій моделі СТМС можна виконувати попередню оцінку стратегій омолодження, враховуючи як інтенсивність використання пристрою, так і стан старіння в ОС Android.

1.6. Висновки до розділу 1.

У даному розділі розглянуто поняття старіння ПЗ та його основні характеристики. Визначено, що явище старіння має значний негативний вплив на якість ПЗ, зокрема, надійність та продуктивність. Враховуючи загальну складність програмних систем, а також, особливості дефектів та помилок старіння, ефективним інструментом протидії негативним ефектам явища старіння є виконання процедури омолодження ПЗ.

Сучасні мобільні та вбудовані пристрої широко використовуються в повсякденному житті в різних аспектах, що визначає необхідність забезпечення високої якості ПЗ. Також, вони є особливо вразливими до ефектів старіння, оскільки тривалий час працюють без перезавантаження та мають обмежену кількість ресурсів, таких як пам'ять. Таким чином, обґрунтовується важливість емпіричного та теоретичного дослідження процесу старіння та методів протидії його ефектам в мобільних ОС, зокрема, Android.

Для теоретичного опису процесу старіння і омолодження ПЗ використовуються аналітичні моделі на основі ланцюгів Маркова з неперервним часом розподілу перебування системи в певному стані в деякий час *t*. Аналітичні моделі можуть бути

розширені та удосконалені шляхом урахування особливостей ОС Android та мобільних пристроїв.

Використання гібридних підходів дослідження старіння ПЗ, що включають переваги аналітичних моделей та методів на основі вимірювань є перспективним напрямком у вивченні старіння ПЗ. Підходи на основі вимірювань метрик системи забезпечують аналітичні моделі реальними статистичними даними, що робить можливим виконання точних обчислень, прогнозування старіння та планування омолодження ПЗ.

Описано поняття метрик старіння та їх важливість для виявлення старіння в системі та застосунках. Важливо вибрати ефективні метрики старіння для конкретної платформи і умов, які будуть з високою достовірністю ідентифікувати наявність чи відсутність процесів старіння.

Крім того, для розуміння природи явища старіння та протидії йому, важливо досліджувати фактори, що мають вплив на процес старіння. Визначено загальний набір факторів (технічні характеристики пристрою, типи застосунків, інтенсивність запуску застосунків, події введення, оперативна пам’ять та пам’ять файлового сховища) характерний для всіх систем і явища старіння в цілому, проте, важливим завданням є визначення специфічних факторів для конкретних систем, зокрема, мобільних.

Виконання процедури омолодження ПЗ полягає у регулярному очищенні стану системи від накопичених помилок старіння. В свою чергу, омолодження ПЗ може мати вплив на взаємодію користувача із системою та окремими застосунками. Тому, урахування «теплого» та «холодного» механізмів процедури омолодження ПЗ в побудові моделей старіння та методів омолодження є важливим для покращення користувацького досвіду, мінімізації часу простою системи та максимізації ефективності процедури омолодження загалом.

Таким чином, вивчення явища старіння ПЗ в мобільних системах, розроблення методів та засобів протидії ефектам старіння з урахуванням особливостей мобільних пристроїв та їх використання є актуальними науково-прикладними задачами, які будуть розв’язуватись в даній роботі.

РОЗДІЛ 2. МОДЕЛІ СТАРІННЯ ТА ОМОЛОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ В ОПЕРАЦІЙНІЙ СИСТЕМІ ANDROID

2.1. Особливості моделей старіння та омолодження програмного забезпечення для мобільної операційної системи Android

Симулювання процесу старіння чи прогнозування часу омолодження з допомогою аналітичних моделей вимагає, щоб побудована модель враховувала якнайбільше факторів та особливостей використання мобільних пристроїв та ОС Android, а також, особливості реалізації механізмів омолодження ПЗ. В цьому розділі пропонуються аналітичні моделі старіння та омолодження, які покращують та розширюють існуючу аналітичну модель [103] на основі ланцюгів Маркова [106] з неперервним часом розподілу для ОС Android, яка враховує як процес старіння та омолодження, так і активність використання мобільного пристрою користувачем. Базова модель, яка використовується в даній роботі для ОС Android є доволі простою, тому, перш за все необхідно виконати її аналіз та аналіз її компонент, визначити основні переваги та виділити недоліки, які потребують покращення та виправлення.

Процес старіння та омолодження в роботі авторів представлений моделлю, яка описує систему, що може перебувати в одному із чотирьох можливих станів: «Young», «Old», «Recovering» та «Rebirth». В даній роботі пропонується використати термін «Failure» замість «Recovering», оскільки він позначає стан відмови системи внаслідок старіння, а відновлення після відмови є наслідком після відмови. А також, замінити «Rebirth» на «Rejuvenation», оскільки цей термін безпосередньо позначає стан в якому відбувається виконання процедури омолодження.

Припускається, що після увімкнення мобільного пристрою система характеризується високою продуктивністю роботи, а імовірність відмов внаслідок старіння є дуже низькою, тобто система перебуває в стані «Young». Після тривалого використання мобільного пристрою користувачем система, внаслідок впливу процесу старіння, з певною інтенсивністю a_{YO} може перейти в стан «Old», який характеризується високою імовірністю виникнення відмов старіння. В залежності від інтенсивності відмов внаслідок старіння a_{OF} можливий перехід зі стану «Old» в стан «Failure», який описує інтервал часу від моменту відмови старіння до

перезавантаження пристрою користувачем чи системою, що дозволяє системі повернутись в стан «Young» з інтенсивністю a_{FY}.

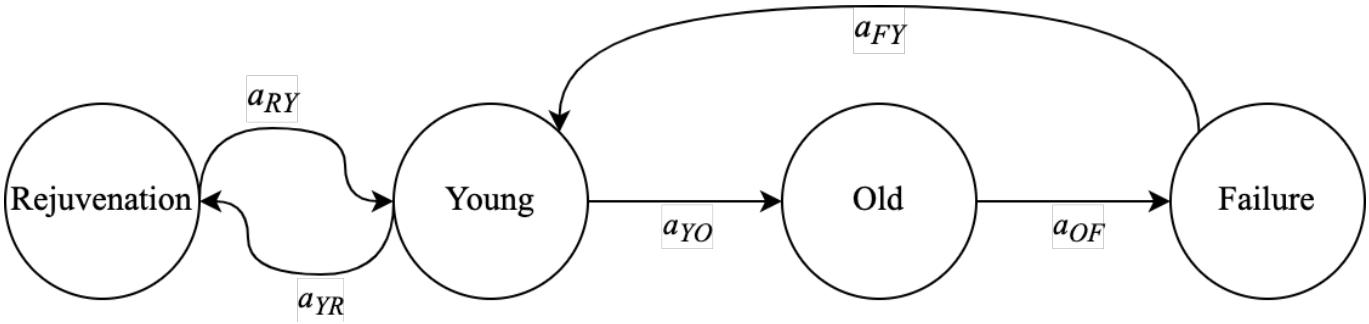


Рис. 2.1. Модель старіння із завчасним виконанням процедури омолодження ПЗ

Виконання процедури омолодження в даній моделі відбувається в стані «Rejuvenation» і, на відміну від простої загальної моделі старіння та омолодження, описаній в попередньому розділі (Рис. 1.2), виконується на ранньому етапі, коли система перебуває в стані «Young», а не в стані «Old». Такий підхід дозволяє моделювати процес старіння, де омолодження виконується з інтенсивністю a_{YR} завчасно до переходу в стан «Old», а прогнозування омолодження на основі такого підходу може покращити характеристики надійності і продуктивності ПЗ, оскільки таким чином зменшується імовірність переходу в стани «Old» та «Failure». Однак, ця модель не враховує умови, які характеризують стан системи «Young». Оскільки в цьому стані система є продуктивною, а процес старіння може не спостерігатись або мати незначний вплив на погіршення продуктивності та збільшення інтенсивності відмов, то виконання процедури омолодження може бути недоцільним в «молодому» стані.

В свою чергу, модель активності використання мобільного пристрою представлена на рис. 2.2. у вигляді простого графу станів «Active» та «Sleep» та відповідних переходів між ними з інтенсивностями a_{AS} та a_{SA}.

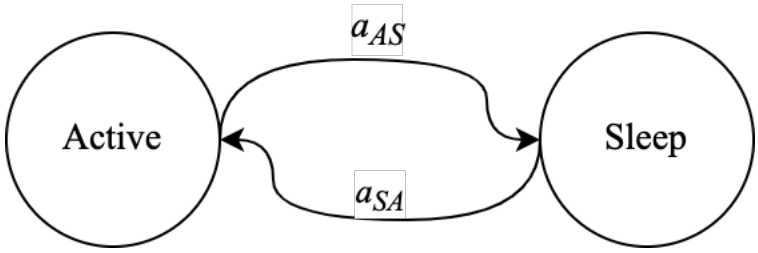


Рис. 2.2. Модель активності використання мобільного пристрою

Мобільний пристрій, в залежності від активності його використання, може перебувати у двох станах:

- Active – користувач активно використовує мобільний пристрій, або виконуються користувацькі процеси;
- Sleep – телефон знаходиться в режимі очікування і робоче навантаження на систему незначне.

Врахування цих станів під час моделювання процесу старіння є важливим з точки зору надійності ПЗ та користувацького досвіду. Виконання процедури омолодження може негативно вплинути на досвід користувача, оскільки може перервати процеси чи застосунки користувача в момент їх активного виконання. Тому, важливим завданням є планування омолодження в той час, коли система з найбільшою імовірністю перебуватиме в стані очікування, тобто в стані «Sleep».

Основним завданням оригінальної моделі (рис. 2.3) старіння та омолодження з урахуванням активності використання мобільного пристрою користувачем є визначення оптимального часу переходу в стан «Rejuvenation», випереджаючи перехід в стан «Old». Важливою умовою вибору оптимального часу в моделі є уникнення виконання омолодження тоді, коли користувач активно використовує мобільний пристрій, тобто система перебуває у стані «Active». Іншими словами, модель передбачає виконання процедури омолодження тільки в стані SmY.

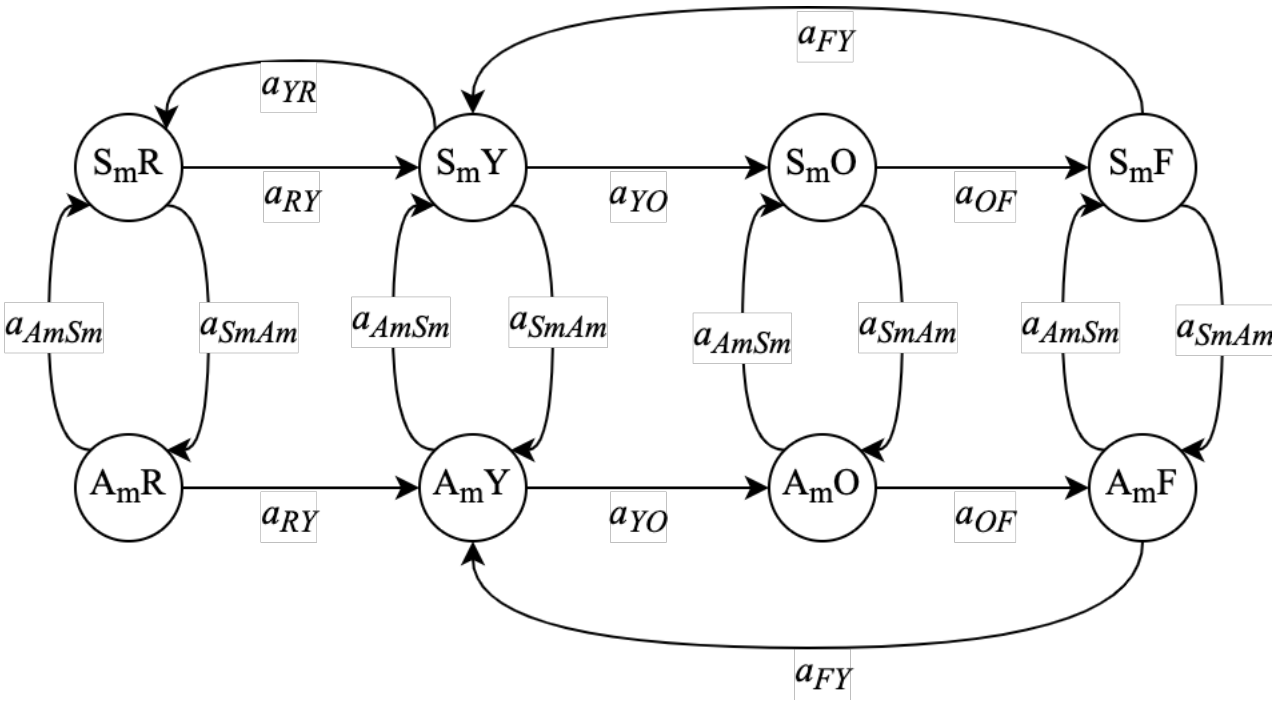


Рис. 2.3. Модель старіння та омолодження з урахуванням активності використання мобільного пристрою

Модель старіння та омолодження з урахуванням активності використання мобільного пристрою не враховує той факт, що виконання процедури омолодження призводить до тимчасової непрацездатності системи, тобто перехід в стан системи AmR, як і перебування системи в цьому стані, може бути неможливим. Можна допустити, що мобільний пристрій може використовуватись користувачем під час виконання «теплого» омолодження при умові, що такий механізм омолодження не перериватиме важливі для користувача процеси. Однак, механізм «теплого» омолодження є менш ефективним за «холодне» омолодження, тому в існуючій моделі виконання «теплого» омолодження в стані «Old» може бути неефективним для повернення в стан «Young». Таким чином, важливим завданням покращення існуючої моделі є урахування «теплого» і «холодного» механізму омолодження як з точки зору ефективності виконання омолодження, так і з точки зору використання мобільного пристрою користувачем.

Оскільки процес старіння та омолодження представлений моделлю на основі ланцюга Маркова, то ймовірність перебування системи в i-му стані $P_i(t)$ можна отримати з розв’язку системи рівнянь Колмогорова-Чепмена:

$$\frac{dP_i(t)}{dt} = -\sum_{j \in S} a_{ij} P_i(t) + \sum_{j \in S} a_{ji} P_j(t), i \in S, \quad (2.1)$$

де $P_i(t)$ ($P_j(t)$) – це ймовірність перебування системи в певний час t в стані $i(j)$, a_{ij} (a_{ji}) – це інтенсивність переходу із стану i в стан j (із стану j в стан i) в певний час t, S – це множина всіх можливих станів системи.

Розв'язок системи диференційних рівнянь (2.1) можна отримати з допомогою чисельних методів, зокрема, в цій роботі використовується метод Рунге-Кутта.

Також, формулу (2.1) можна представити у матричному вигляді для обчислення вектору ймовірностей перебування системи в кожному із можливих станів у довільний час t:

$$P(t) = P(0) \cdot \exp(Q \cdot t), \quad (2.2)$$

де Q – це матриця інтенсивностей переходів між станами, а $P(0)$ – це початковий вектор імовірностей.

Оскільки розподіл часу в моделях на основі ланцюгів Маркова з неперервним часом є експонентним, то середня тривалість переходів між станами обчислюється за формулою:

$$T_{avg} = \frac{1}{a}, \quad (2.3)$$

де a – це інтенсивність переходу між двома довільними станами системами.

Таким чином, в даній роботі пропонуються моделі, які ураховують різні чинники та умови старіння особливі для ОС Android:

- модель старіння та омолодження з урахуванням різних рівнів старіння та активності використання мобільного пристрою;
- модель старіння та омолодження з урахуванням різних рівнів заряду батареї та активності використання мобільного пристрою;
- комплексна модель з урахуванням різних рівнів старіння, заряду батареї та активності використання мобільного пристрою.

Аналітичні моделі процесу старіння та омолодження представлені у вигляді ланцюгів Маркова з неперервним часом розподілу. Кроки, виконані для побудови моделей та їх оцінки, наступні:

- визначено можливі стани системи, які описують процес старіння та омолодження з урахування тих чи інших чинників і умов;

- визначено можливі переходи між станами та інтенсивності цих переходів;

- побудовано графічне представлення моделі у вигляді графу станів та переходів між ними;

- описано аналітичне представлення моделі у вигляді системи диференційних рівнянь на основі формули (2.1);

- визначено набори тестових інтенсивностей для симуляції процесу старіння та омолодження;

- визначено розв’язки для систем диференційних рівнянь відповідних аналітичних моделей з допомогою методу Рунге-Кутта для визначених наборів тестових симуляцій;

- виконано аналіз розв’язків для оцінки ефективності запропонованих моделей.

2.2. Модель з урахуванням різних рівнів старіння та стратегій виконання омолодження програмного забезпечення

Процес старіння характеризується поступовим погіршенням продуктивності, що призводить до відмови старіння. Враховуючи цей фактор в побудові моделей важливо розрізняти рівні старіння системи, що дозволить будувати різні стратегії омолодження ПЗ та застосовувати відповідні механізми омолодження для того чи іншого рівня. Наприклад, при критично низькому рівні продуктивності та високій імовірності відмов старіння необхідно виконувати повне перезавантаження мобільного пристрою, при помірному рівні продуктивності можна виконувати перезавантаження окремих застосувань чи сервісів системи, а при відсутності наявних процесів старіння не планувати виконання омолодження зовсім. Крім того, урахування рівнів старіння дозволяє використати переваги методів виявлення старіння на основі вимірювань та порогових значень метрик, що дає більш точні відомості про стан системи у конкретних умовах використання мобільного пристрою.

Пропонується розширити модель старіння із завчасним виконанням омолодження таким чином, щоб процедура омолодження не виконувалась в

молодому стані системи. В цьому випадку, загальна модель старіння та омолодження із різними рівнями старіння може бути представлена набором наступних п’яти можливих станів системи: «Young», «Aging», «Old», «Rejuvenation», «Failure», де «Young», «Aging» та «Old» відображають поступове погіршення продуктивності системи в контексті старіння ПЗ. Таким чином, загальна розширена модель може бути представлена у вигляді графу станів та переходів (Рис. 2.4), де її стани мають наступні характеристики:

- Young – це стан системи, яких характеризується високим рівнем продуктивності та низьким рівнем використання системних ресурсів, наприклад, вимірювані метрики старіння не перевищують визначені порогові значення;
- Aging – це стан системи в якому спостерігається погіршення продуктивності та збільшення використання системних ресурсів, але цей процес ще не має значного впливу на досвід користувача, а вимірювані метрики старіння знаходяться в межах порогових значень переходу в стан «Old»;
- Old – це стан системи, коли користувач відчуває значні затримки в графічному інтерфейсі, спостерігається виснаження системних ресурсів, що може призводити до збоїв у роботі користувацьких застосунків;
- Rejuvenation – це стан системи в якому виконується процедура омолодження, система може бути тимчасово недоступною у випадку виконання «холодної» процедури омолодження;
- Failure – це стан відмови внаслідок старіння в якому відбувається відновлення системи до стану «Young». Відновлення в цьому випадку може бути шляхом перезавантаженням мобільного пристрою користувачем, або самою системою.

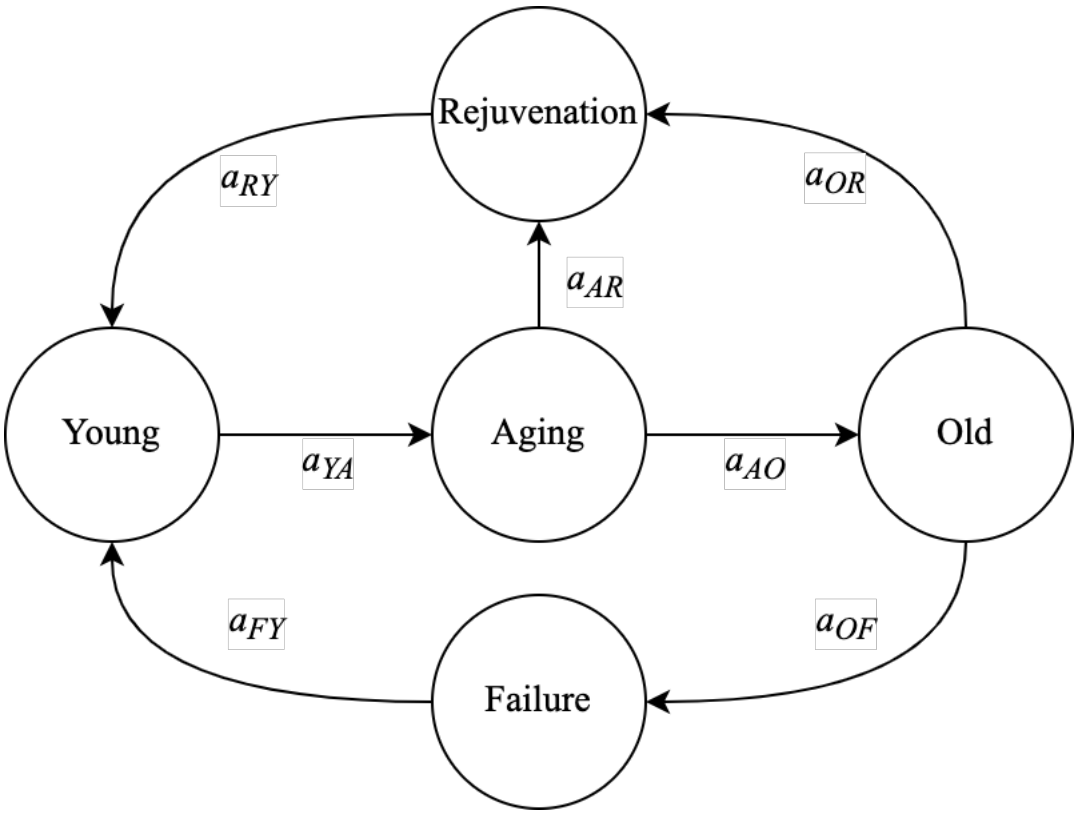


Рис. 2.4. Граф станів та переходів загальної моделі старіння та омолодження, що враховує різні рівні старіння

Для обчислення інтенсивностей переходів a_{ij}, так само як і часу переходу T_{ij} між станами важливо враховувати наявність чи відсутність постійного тренду погіршення метрик в таких станах як «Young», «Aging» та «Old». Наприклад, в системі в станах «Young» або «Aging» може не спостерігатись збільшення трендів FDT, тому перехід в наступні стани «Aging» та «Old» може не відбутись. В цьому випадку виконання процедури омолодження, так само як і необхідність її прогнозування, може бути недоцільним.

Таким чином, на відміну від оригінальної моделі [103], додатковий стан «Aging» дозволяє чітко виділити процес старіння в системі, який може призвести до стану «Old». В цьому випадку розширена модель дозволяє виконувати омолодження в двох станах «Aging» та «Old», що забезпечується переходами з інтенсивністю a_{AR} та a_{OR} відповідно.

Об’єднавши модель старіння та омолодження, що враховує різні рівні старіння із моделлю активності використання мобільного пристрою користувачем можна

отримати покращену модель для мобільної ОС [5], яку представлено у вигляді графу станів та переходів на рис. 2.5. Можливі стани системи моделі, що враховує різні рівні старіння та активність використання мобільного пристрою користувачем:

- AmY – це стан активного використання пристрою з високим рівнем продуктивності;
- AmA – це стан активного використання пристрою в якому спостерігається процес старіння;
- AmO – це стан активного використання пристрою з низьким рівнем продуктивності;
- AmR – це стан виконання процедури омолодження під час активного використання пристрою;
- AmF – це стан відмови внаслідок старіння під час активного використання пристрою;
- SmY – це стан очікування пристрою з високим рівнем продуктивності;
- SmA – це стан очікування пристрою в якому спостерігається процес старіння;
- SmO – це стан очікування пристрою з низьким рівнем продуктивності;
- SmR – це стан виконання процедури омолодження під час очікування пристрою;
- SmF – це стан відмови внаслідок старіння під час очікування пристрою.

Планування омолодження ПЗ полягає у виборі оптимального часу його виконання. Оптимальним часом з точки зору користувача є ті стани системи, коли мобільний пристрій не використовується активно, а саме, SmY, SmA, SmO. В свою чергу оптимальним часом з точки зору мобільного пристрою є стани системи, коли спостерігається погіршення продуктивності, процедура омолодження виконується до настання відмови внаслідок старіння, а також, омолодження не перериває роботу процесів з високим пріоритетом, тобто в станах SmA та SmO. Таким чином, оптимальним часом омолодження для користувача та мобільного пристрою є стани SmA та SmO.

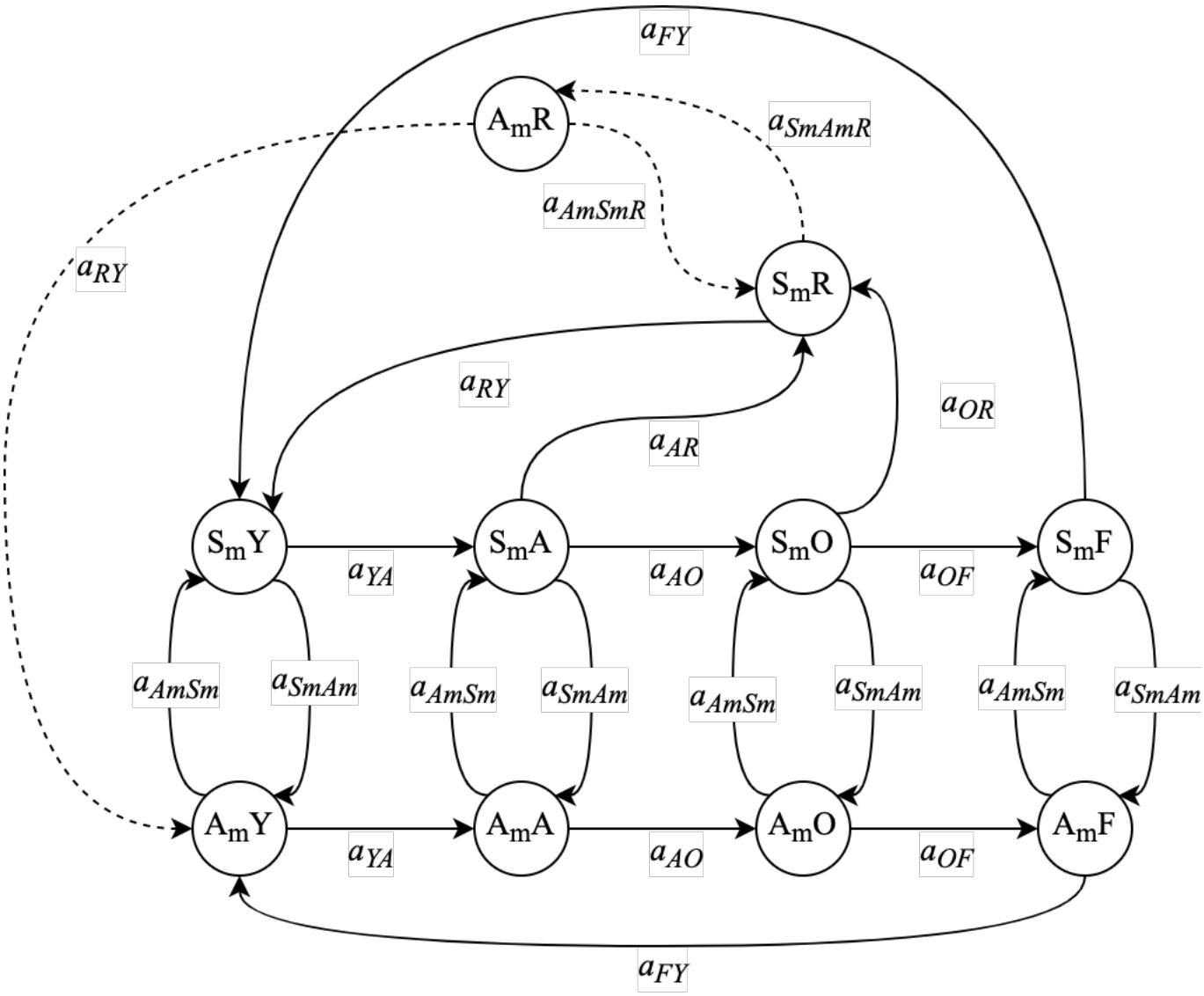


Рис. 2.5. Граф станів та переходів моделі старіння та омолодження, що враховує різні рівні старіння та активність використання мобільного пристрою користувачем

Запропонована модель дозволяє врахувати три основні стратегії планування омолодження, а саме:

1. Одночасне омолодження із станів «Aging» та «Old», тобто тоді, коли $a_{AR} > 0.0$ та $a_{OR} > 0.0$;

2. Омолодження тільки із стану «Aging», тобто тоді, коли $a_{AR} > 0.0$ та $a_{OR} = 0.0$;

3. Омолодження тільки із стану «Old», тобто тоді, коли $a_{AR} = 0.0$ та $a_{OR} > 0.0$.

Враховуючи існування різних механізмів омолодження, а саме «холодного» та «теплого» омолодження, в модель введено інтенсивності переходів між станами AmR та SmR. Ці параметри дозволяють визначити тип процедури омолодження. При умові,

коли $a_{SmAmR} = 0$ та $a_{AmSmR} = 0$, виконується «холодне» омолодження, тобто система знаходиться в стані простою, тому перехід в стан активного використання користувачем не можливий. Таким чином, враховуючи активність використання мобільного пристрою користувачем, важливим завданням є вибір оптимального механізму омолодження, який забезпечуватиме високий рівень працездатності системи.

Аналітичне представлення моделі у вигляді системи диференційних рівнянь Колмогорова-Чепмана:

$$
\begin{aligned}
\frac{dP_{SmY}(t)}{dt} &= -(a_{YA} + a_{SmAm})P_{SmY}(t) + a_{FY}P_{SmF}(t) + a_{AmSm}P_{AmY}(t) + a_{RY}P_{SmR}(t);\\
\frac{dP_{SmA}(t)}{dt} &= -(a_{AO} + a_{SmAm} + a_{AR})P_{SmA}(t) + a_{YA}P_{SmY}(t) + a_{AmSm}P_{AmA}(t);\\
\frac{dP_{SmO}(t)}{dt} &= -(a_{OF} + a_{OR} + a_{SmAm})P_{SmO}(t) + a_{AO}P_{SmA}(t) + a_{AmSm}P_{AmO}(t);\\
\frac{dP_{SmF}(t)}{dt} &= -(a_{FY} + a_{SmAm})P_{SmF}(t) + a_{OF}P_{SmO}(t) + a_{AmSm}P_{AmF}(t);\\
\frac{dP_{AmY}(t)}{dt} &= -(a_{YA} + a_{AmSm})P_{AmY}(t) + a_{FY}P_{AmF}(t) + a_{RY}P_{AmR}(t) + a_{SmAm}P_{SmY}(t);\\
\frac{dP_{AmA}(t)}{dt} &= -(a_{AO} + a_{AmSm})P_{AmA}(t) + a_{YA}P_{AmY}(t) + a_{SmAm}P_{SmA}(t);\\
\frac{dP_{AmO}(t)}{dt} &= -(a_{OF} + a_{AmSm})P_{AmO}(t) + a_{AO}P_{AmA}(t) + a_{SmAm}P_{SmO}(t);\\
\frac{dP_{AmF}(t)}{dt} &= -(a_{FY} + a_{AmSm})P_{AmF}(t) + a_{OF}P_{AmO}(t) + a_{SmAm}P_{SmF}(t);\\
\frac{dP_{SmR}(t)}{dt} &= -(a_{RY} + a_{SmAmR})P_{SmR}(t) + a_{AR}P_{SmA}(t) + a_{OR}P_{SmO}(t) + a_{AmSmR}P_{AmR}(t);\\
\frac{dP_{AmR}(t)}{dt} &= -(a_{RY} + a_{AmSmR})P_{AmR}(t) + a_{SmAmR}P_{SmR}(t).
\end{aligned}
\tag{2.4}
$$

Для аналізу і верифікації запропонованої моделі старіння та омолодження визначено розв’язки для системи диференційних рівнянь методом Рунге-Кутта четвертого порядку. Середній час переходу між станами системи є наближеними значеннями, які дозволяють симулювати процес старіння та омолодження для виконання експериментальних обчислень. В додатку А представлено набір тестових

даних для симуляції в різних умовах старіння, активності використання мобільного пристрою та типів процедури омолодження.

Активність використання мобільного пристрою представлена двома наборами інтенсивностей переходів:

- Помірне використання мобільного пристрою, де T_{SmAm} = 75 хв. (a_{SmAm} = 0.013), а T_{AmSm} = 10 хв. (a_{AmSm} = 0.1);
- Активне використання, де T_{SmAm} = 30 хв. (a_{SmAm} = 0.033), а T_{AmSm} = 50 хв. (a_{AmSm} = 0.02).

Модель старіння ПЗ описує два пристрої з різним рівнем продуктивності та швидкістю старіння:

- Пристрій з високим рівнем продуктивності та стійкістю до старіння ПЗ, де T_{YA} = 48 год. (a_{YA} = 0.00035), T_{AO} = 96 год. (a_{AO} = 0.00018), T_{OF} = 24 год. (a_{OF} = 0.00069), та T_{FY} = 1 хв. (a_{FY} = 1.0);
- Пристрій з низьким рівнем продуктивності та вразливістю до старіння ПЗ, де T_{YA} = 4 год. (a_{YA} = 0.0042), T_{AO} = 8 год. (a_{AO} = 0.0021), T_{OF} = 2 год. (a_{OF} = 0.0083), та T_{FY} = 5 хв. (a_{FY} = 0.2).

В цій роботі розглядається два варіанти відмов старіння ПЗ, а саме, порівняння симуляцій із відновленням після відмови старіння та без відновлення:

- Відмова старіння спричиняє перезавантаження, яке повертає мобільний пристрій в стан «Young», тобто система перебуває в стані FY тільки той час, який необхідний для виконання перезавантаження;
- Відмова старіння не повертає систему в стан «Young», тобто, T_{FY} = 0 хв. (a_{FY} = 0.0).

Стратегії омолодження представлені двома наборами даних для двох мобільних пристроїв з різними інтенсивностями старіння.

Процедура омолодження для пристрою з високим рівнем продуктивності та стійкістю до старіння триває в середньому одну хвилину (a_{RY} = 1.0), а стратегії планування представлено наступним набором значень T_{avg} та a_{ij}:

- Омолодження в станах «Aging» та «Old», де T_{AR} = 48 год. (a_{AR} = 0.00035), T_{OR} = 12 год. (a_{OR} = 0.0014);

- Омолодження тільки в стані «Aging», де T_{AR} = 48 год. (a_{AR} = 0.00035), T_{OR} = 0 год. (a_{OR} = 0.0);

- Омолодження тільки в стані «Old», де T_{AR} = 0 год. (a_{AR} = 0.0), T_{OR} = 12 год. (a_{OR} = 0.0014);

- Омолодження виконується із затримкою в стані «Aging» та «Old», де T_{AR} = 144 год. (a_{AR} = 0.00012), T_{OR} = 36 год. (a_{OR} = 0.00046).

Процедура омолодження для пристрою з низьким рівнем продуктивності та вразливістю до старіння триває в середньому 5 хвилин (a_{RY} = 0.2), а планування стратегій представлено наступним набором значень T_{avg} та a_{ij}:

- Омолодження в станах «Aging» та «Old», де T_{AR} = 4 год. (a_{AR} = 0.0042), T_{OR} = 1 год. (a_{OR} = 0.017);

- Омолодження тільки в стані «Aging», де T_{AR} = 4 год. (a_{AR} = 0.0042), T_{OR} = 0 год. (a_{OR} = 0.0);

- Омолодження тільки в стані «Old», де T_{AR} = 0 год. (a_{AR} = 0.0), T_{OR} = 1 год. (a_{OR} = 0.017);

- Омолодження виконується із затримкою в стані «Aging» та «Old», де T_{AR} = 12 год. (a_{AR} = 0.0014), T_{OR} = 3 год. (a_{OR} = 0.0056).

Для перевірки запропонованої моделі та оцінки процесу старіння виконано симуляції SIM-0 - SIM-4 (див. Додаток А) в яких не виконується процедура омолодження. Вплив процесу старіння на UX може бути оцінений з допомогою значень $P_{AmY}(t)$, $P_{AmA}(t)$, $P_{AmO}(t)$ і $P_{AmF}(t)$, тобто з допомогою імовірностей перебування системи на різних рівнях старіння під час активного використання мобільного пристрою користувачем. В свою чергу, для оцінки впливу відновлення після відмови старіння виконано симуляцію SIM-0, яка є аналогічна до SIM-1, за винятком переходу із стану «Failure» в стан «Young» (a_{FY} = 0.0). Тобто, відмова старіння є термінальним непрацездатним станом системи, а відновлення може відбутись в невизначений час у майбутньому.

В симуляціях SIM-1 (рис. 2.6) та SIM-0 (рис. 2.7) неактивний користувач використовує мобільний пристрій, що старіє повільно. Порівняння цих симуляцій дозволяє охарактеризувати вплив відновлення після старіння, а також оцінити

важливість застосування процедури омолодження для покращення UX та збільшення часу працездатності мобільного пристрою.

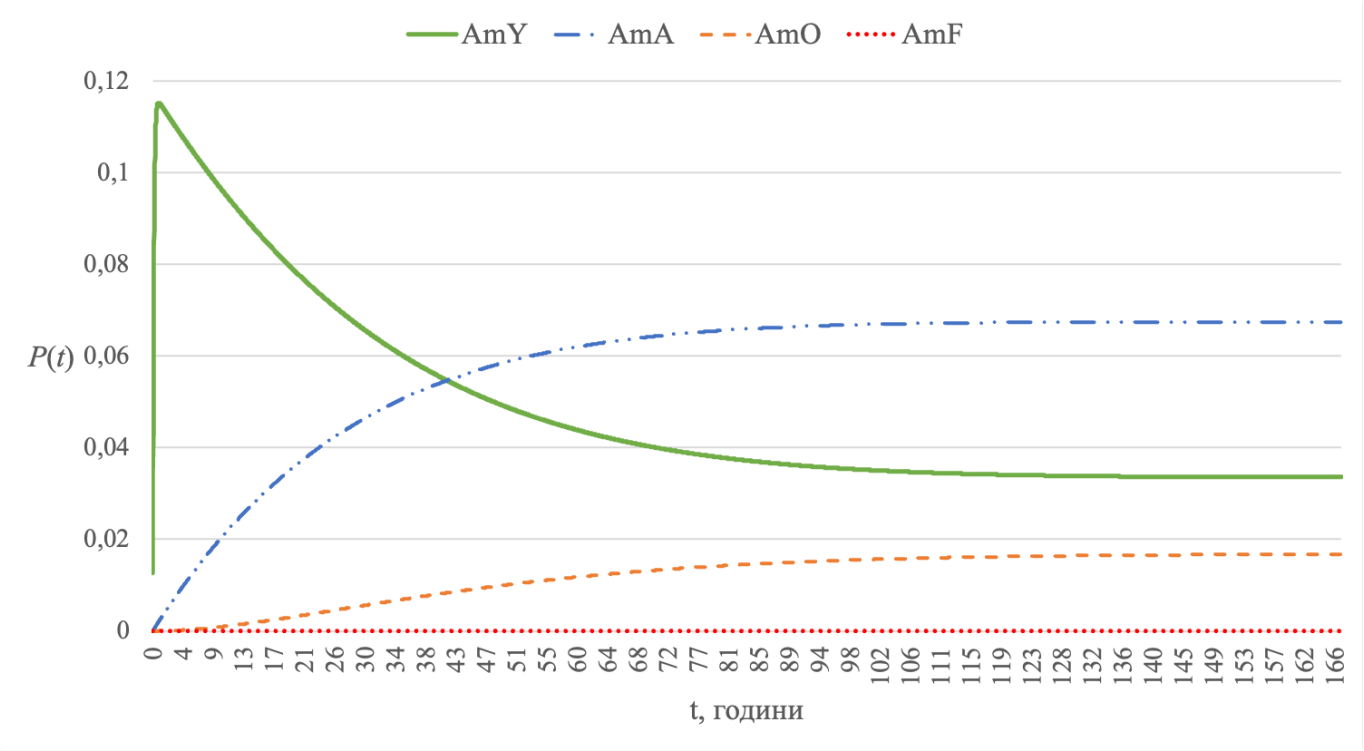


Рис. 2.6. Симуляція процесу старіння ПЗ без омолодження з відновленням після відмови старіння

Ймовірності перебування в усіх станах симуляції SIM-1 (рис. 2.6) досягають стійкого рівня після 100 годин використання мобільного пристрою. Така поведінка моделі пояснюється наявністю стану «Failure» в якому відбувається відновлення з поверненням у стан «Young». У випадку SIM-1, мобільний пристрій обов'язково прийде до, так би мовити, «плато відновлення», яке забезпечує повернення пристрою в працездатний стан, але змушує користувача чи систему виконувати термінове перезавантаження мобільного пристрою для відновлення працездатності, що має негативний вплив на отримання позитивного досвіду користувача.

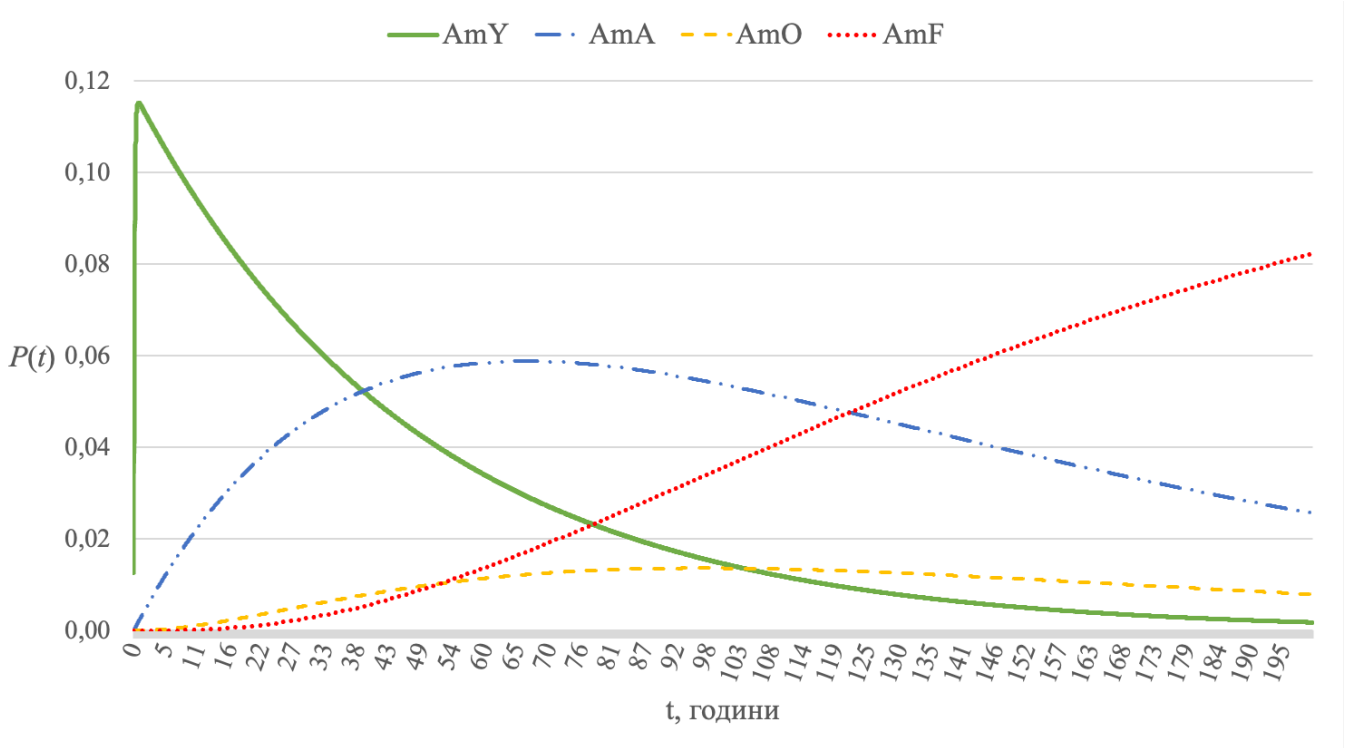


Рис. 2.7. Симуляція процесу старіння ПЗ без омолодження та без відновлення після відмови старіння

Порівняння симуляцій SIM-0 та SIM-1 дозволяє зробити висновок, що без будь якої реакції на відмову старіння (рис. 2.7), ймовірність перебування в молодому стані AmY прямує до нуля, а процес старіння (тобто, зміни станів AmY, AmA, AmO та AmF) призводить до повної відмови системи. Отже, аналіз виконаних симуляцій показує, що важливим завданням виконання процедури омолодження для покращення UX є:

- Збільшення ймовірності перебування в працездатному стані з високим рівнем продуктивності: $max(P_{AmY}(t))$;
- Відтермінування процесу старіння та відмов старіння: $min(P_{AmA}(t) + P_{AmO}(t) + P_{AmF}(t))$.

Для аналізу стратегій планування омолодження виконано симуляції старіння різних комбінацій мобільних пристроїв та активності користувачів SIM-5 - SIM-20 (див. Додаток А). Виконано порівняння отриманих результатів із симуляціями без виконання омолодження (SIM-1 - SIM-4). Ефективність процедури омолодження в різних стратегіях може бути оцінена за допомогою ймовірності перебування системи

в стані AmY. Тобто, ефективна стратегія омолодження має забезпечувати найбільшу ймовірність перебування мобільного пристрою в стані з високою продуктивністю під час його активного використання.

На рисунках 2.8-2.11 показано порівняння ефективності різних стратегій омолодження для чотирьох наборів симуляцій. Кожен набір описує різні стратегії для однакових умов старіння. Криві на графіках пронумеровані від 1 до 5 в порядку спадання їх ефективності.

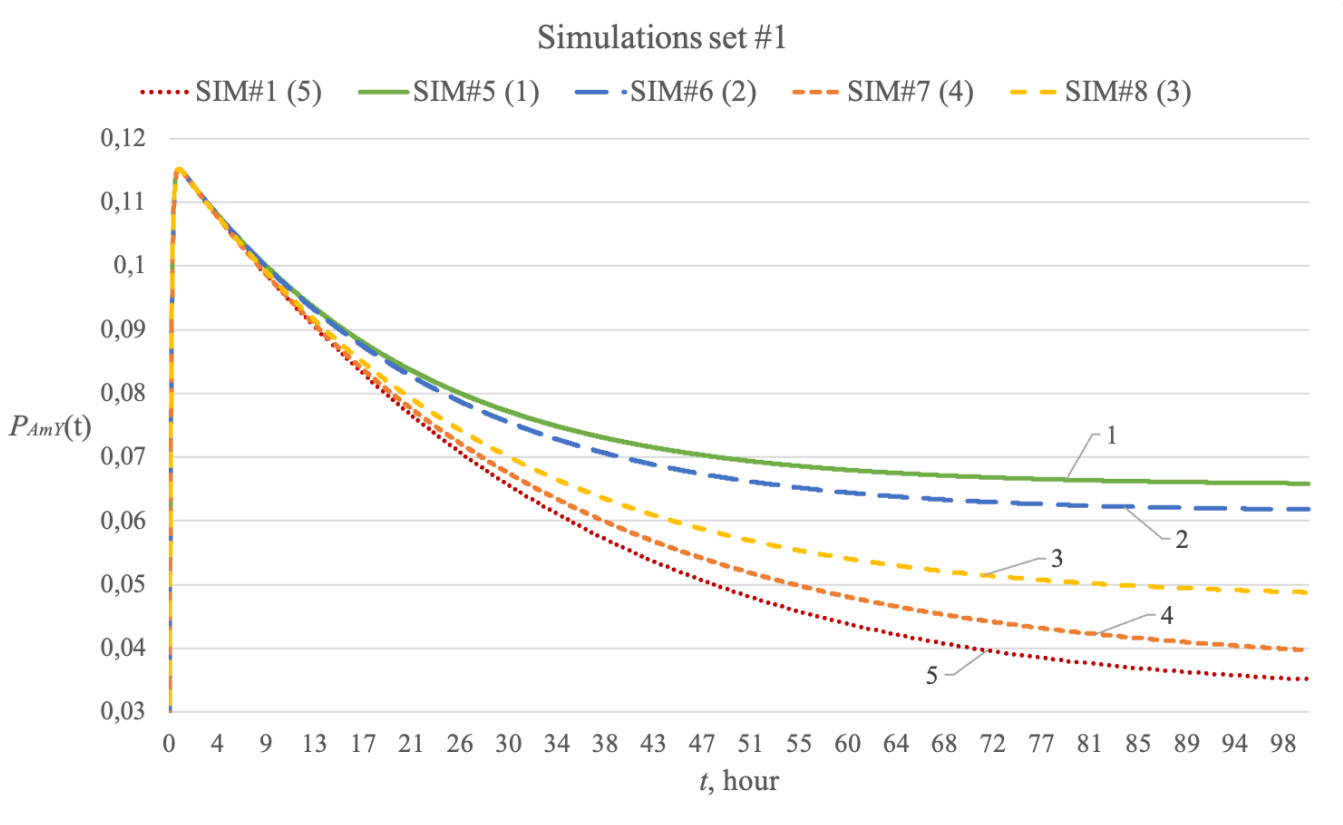


Рис. 2.8. Ефективність різних стратегій омолодження для набору симуляцій №1

В усіх випадках виконання омолодження як із стану «Aging», так і з стану «Old» дозволяє забезпечити найвищий рівень перебування пристрою в стані «Young» (криві під номером 1). Менш ефективною стратегією є планування омолодження тільки зі стану «Aging» (криві під номером 2). Найгіршою стратегією є планування омолодження із стану «Old» (криві під номером 4). Результат моделювання без виконання омолодження очікувано показує найгірший результат (криві під номером 5). Також, можна побачити, що симуляції в яких омолодження заплановане із

затримками для обох станів «Aging» та «Old» (криві під номером 3) показують кращі результати, ніж без омолодження, або можуть бути кращими за планування тільки в стані «Old».

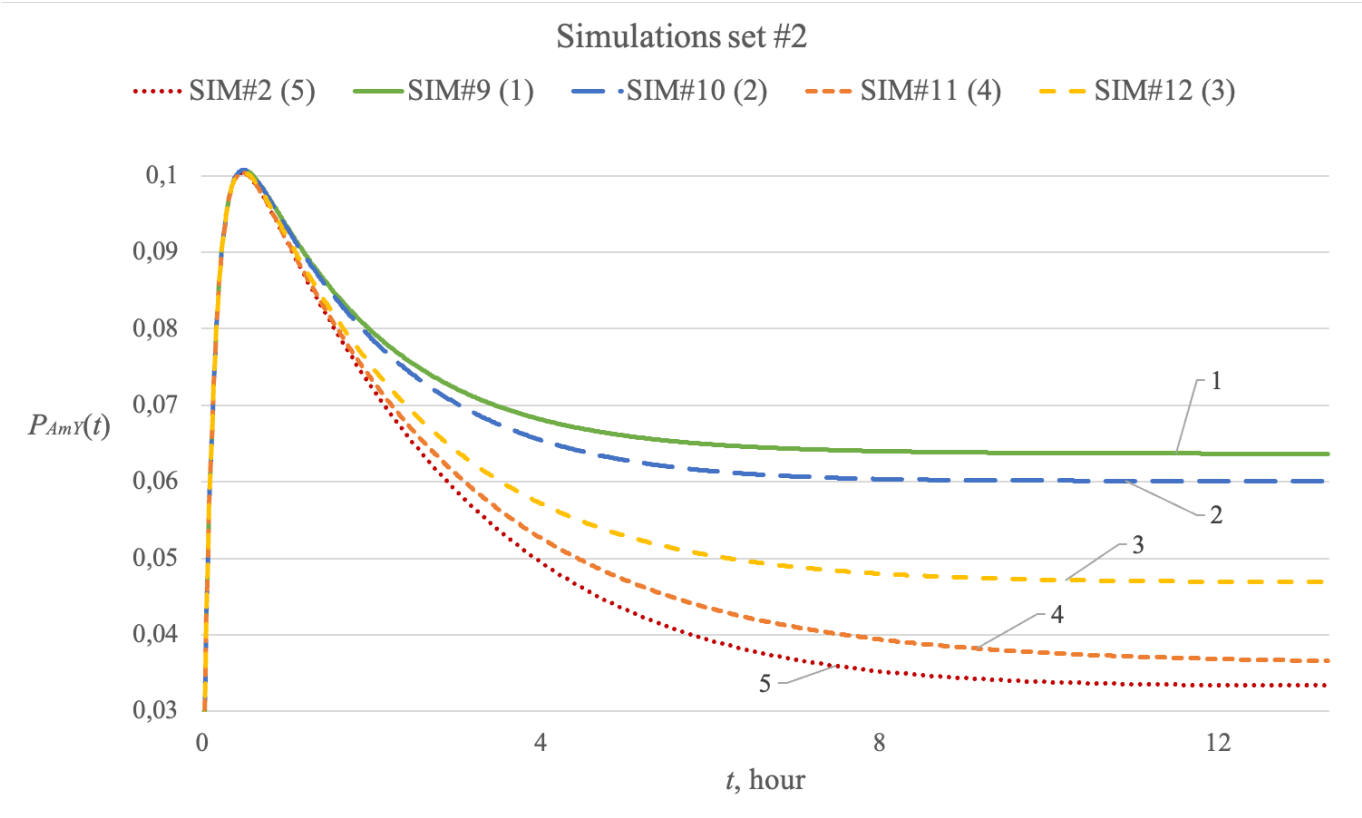


Рис. 2.9. Ефективність різних стратегій омолодження для набору симуляцій №2

Набори симуляцій №1 (рис. 2.8) і №2 (рис. 2.9) описують помірне використання мобільних пристроїв користувачем, тому в обох випадках імовірність перебування в стані AmY не може перевищувати 11-12% і з часом знижується до значень в межах 3,5-6,5%. Симуляція старіння із омолодженням в стані «Aging» дозволяє збільшити $P_{AmY}(t)$ в 1,77 рази порівняно із симуляцією без виконання омолодження, тобто, із 3,5% до 6,2% на інтервалах часу, де спостерігається так зване «плато» необхідності відновлення після відмови старіння (після 98 і 12 годин використання мобільного пристрою відповідно).

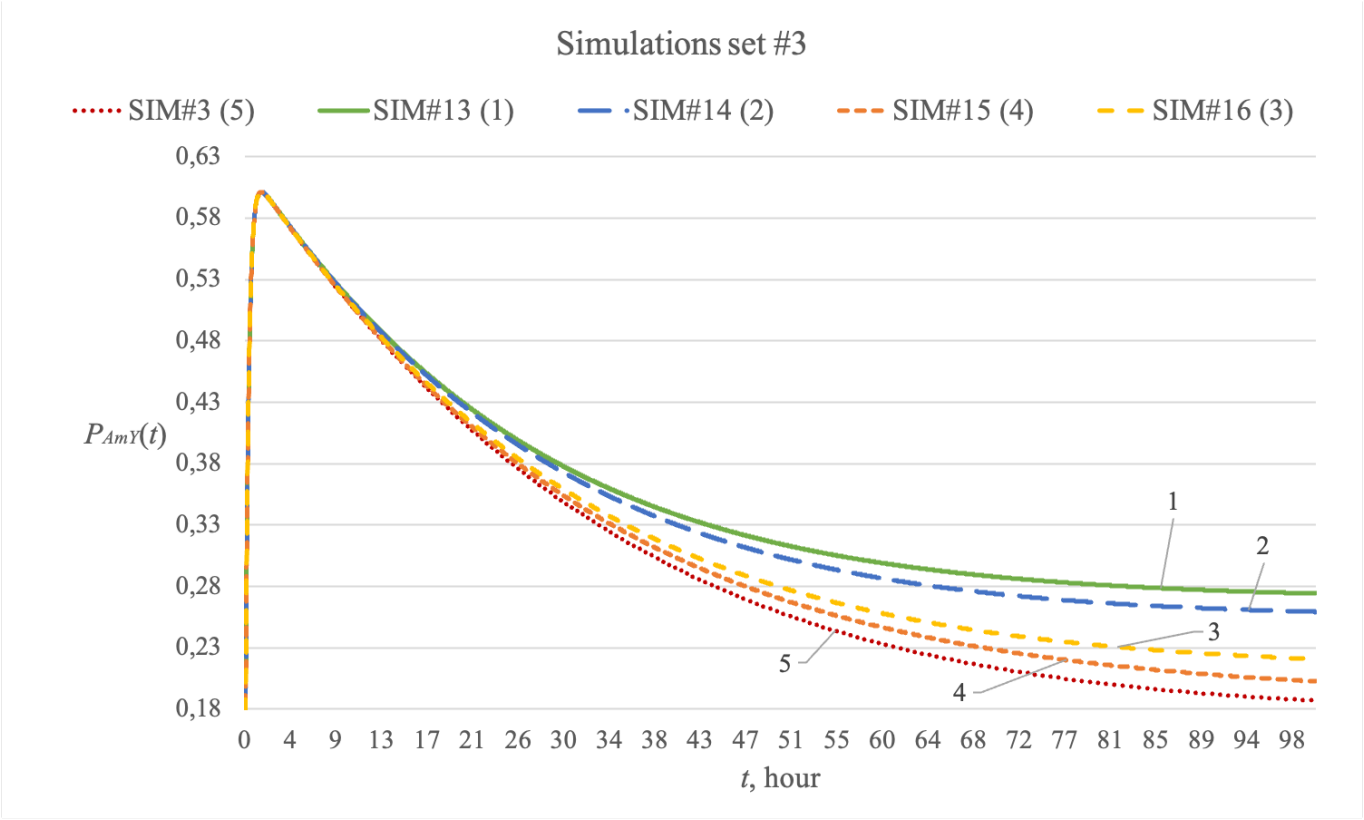


Рис. 2.10. Ефективність різних стратегій омолодження для набору симуляцій №3

Набори симуляцій №3 (рис. 2.10) і №4 (рис. 2.11) описують активне використання мобільних пристроїв користувачем, тому в обох випадках імовірність перебування в стані AmY може перевищувати 50% і з часом знижується до значень в межах 18-27%. Симуляція старіння із омолодженням в стані «Aging» дозволяє збільшити $P_{AmY}(t)$ майже в 1,37-1,47 рази порівняно із симуляцією без виконання омолодження, тобто, із 19% до 26% в наборі симуляцій №3 і з 17% до 25% в наборі симуляцій №4 на інтервалах часу, де спостерігається вихід на так зване «плато» необхідності відновлення після відмови старіння (після 12 і 98 годин використання мобільного пристрою відповідно).

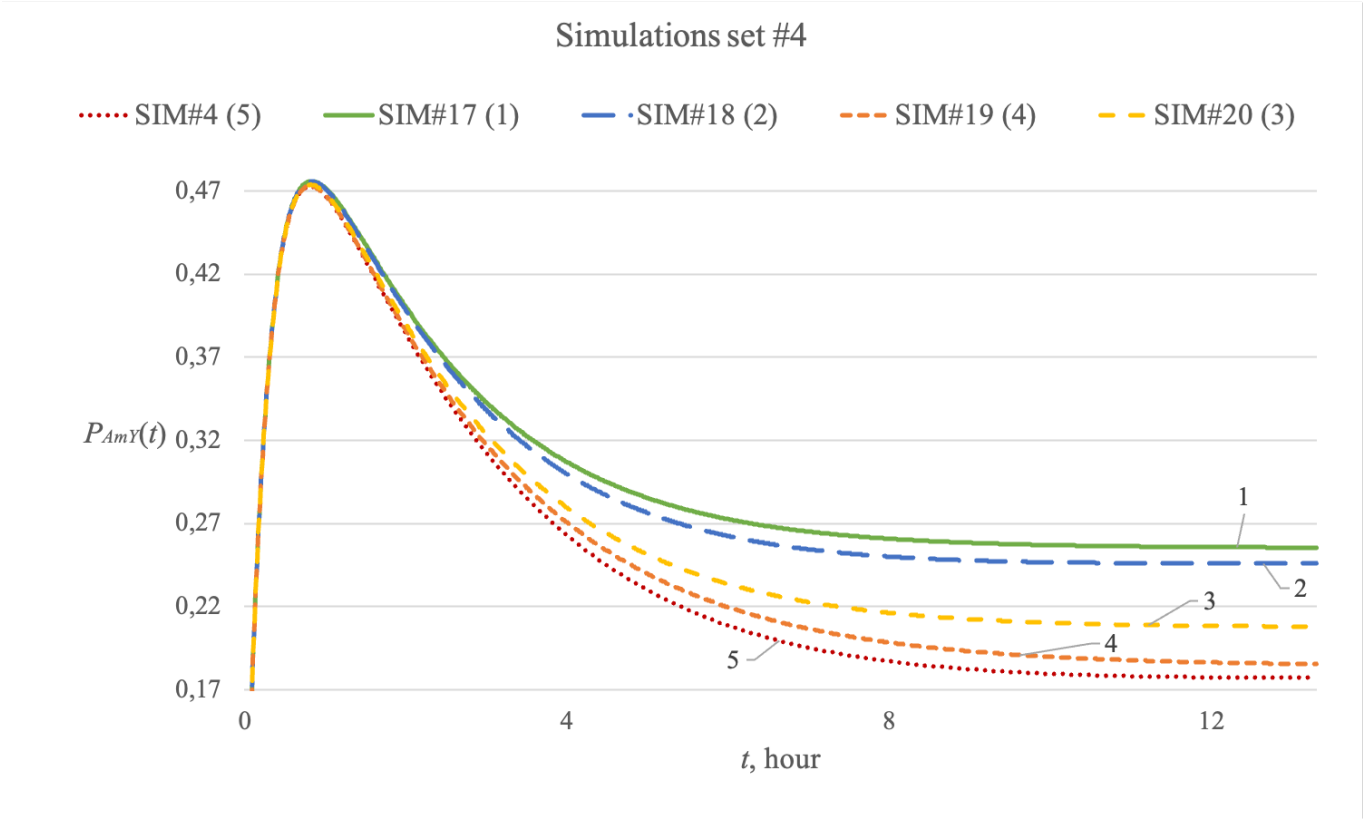


Рис. 2.11. Ефективність різних стратегій омолодження для набору симуляцій №4

Отримані результати підтверджують ефективність завчасного омолодження в стані «Aging» до переходу в стан «Old». В цьому випадку для забезпечення високого рівня працездатності мобільного пристрою потрібно повторно виконувати процедуру омолодження. Хоча омолодження із запізненням (криві під номером 3) збільшує можливість перебування в стані «Young», однак цей випадок не оптимальний, тому важливо вибрати такий час омолодження, який би попереджав виникнення відмови старіння, але не спричиняв необхідність надлишкового виконання омолодження.

Для аналізу механізмів «холодного» омолодження виконано симуляції SIM-21 - SIM-24 (див. Додаток А), де $T_{SmAmR} = T_{AmSmR} = 0.0$. В свою чергу, симуляції SIM-1 - SIM-20 описують процедуру «теплого» омолодження, де $T_{SmAmR} = T_{SmAm}$, $T_{AmSmR} = T_{AmSm}$. Порівняння симуляцій SIM-24 та SIM-18 з однаковими умовами старіння, але різними механізмами омолодження показано на рис. 2.12. Крива під номером 3 показує відсоток збільшення ймовірності перебування системи в стані «Young», якщо виконується процедура «теплого» омолодження.

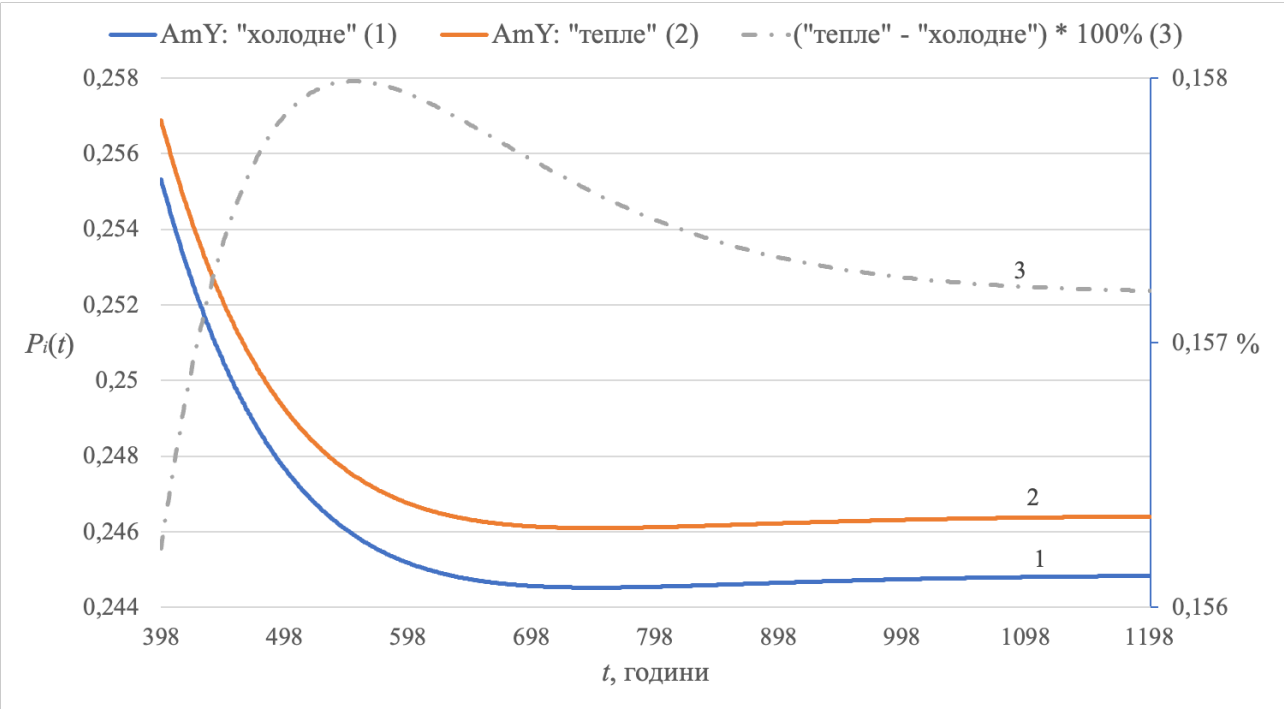


Рис. 2.12. Порівняння механізмів «теплого» та «холодного» омолодження на прикладі симуляцій SIM-24 та SIM-18

У випадку «теплого» омолодження (крива 2 на рис. 2.12) мобільний пристрій справді з більшою імовірністю може перебувати в стані AmY, ніж у випадку «холодного» омолодження (крива 1 на рис. 2.12). Симуляції SIM-24 та SIM-18 описують випадок коли активний користувач використовує мобільний пристрій, який швидко старіє, тому важливими завданнями омолодження є забезпечення високого рівня працездатності, а також ефективності вибраної стратегії. «Тепле» омолодження може забезпечити вищий рівень працездатності, однак, якщо пристрій виконує цей тип омолодження в стані «Old», тоді ймовірність відновлення і повернення до стану «Young» може бути низькою у зв'язку із особливостями реалізації цього механізму. Таким чином, запропонована модель старіння з «теплим» омолодженням не може бути застосована для випадку виконання омолодження в стані «Old» без припущення, що ця техніка забезпечить повернення в стан «Young». Іншими словами, можна припустити, що техніка «теплого» омолодження в запропонованій моделі може бути ефективною у випадку її планування в стані «Aging», а «холодне» омолодження у обох станах «Aging» та «Old».

Практична значущість використання додаткових рівнів старіння ПЗ в моделі полягає в можливості застосування методів на основі вимірювань і порогових

значеннях метрик старіння. Результати моделювання показали, що ефективною стратегією омолодження програмного забезпечення є його багаторазове виконання в стані «Old». Запропонована модель дозволяє проводити омолодження, коли це необхідно, тобто коли в системі спостерігається старіння ПЗ після стану «Young». У той же час модель дозволяє планувати омолодження до переходу в низькопродуктивний стан, при якому є більша ймовірність відмови, пов'язаної зі старінням, тобто до «Old» стану. Модель дозволяє підібрати оптимальний механізм омолодження для конкретних умов використання пристрою, що має практичне значення для покращення UX. Таким чином, запропонована модель дозволяє враховувати UX особливості та визначати показники надійності ПЗ.

2.3. Модель старіння та омолодження програмного забезпечення з урахуванням рівня заряду батареї

Особливістю мобільних пристроїв, що відрізняє їх від інших програмно-апаратних систем, є залежність від заряду батареї. Цей чинник є вагомим як для користувача, так і для планування та виконання омолодження ПЗ. Для користувача важливим є збільшення часу роботи пристрою від заряду батареї, що може бути забезпечено шляхом зменшення навантаження на апаратні ресурси системи, зокрема з допомогою виконання процедури омолодження. В свою чергу, для омолодження ПЗ важливо враховувати можливість його виконання в запланований час та ефективність цієї процедури з довготерміновим ефектом. Без урахування заряду батареї заплановане омолодження ПЗ може не відбутись в той час, коли пристрій повністю розряджений. Також варто врахувати, що омолодження ПЗ в пристрої з низьким зарядом може не мати довготермінового ефекту, а навпаки, спричиняти збільшення часу простою системи, що негативно впливатиме на UX.

Таким чином, в моделі старіння та омолодження пропонується врахувати фактор рівня заряду батареї [7], який би дозволив уникнути виконання омолодження в стані з низьким зарядом або повністю розрядженим пристроєм. Перш за все, необхідно описати графічну модель станів заряду батареї мобільного пристрою без урахування процесу старіння та омолодження, що дозволить зробити аналіз процесу

заряду батареї та охарактеризувати нові стани системи. Загальна модель заряду батареї може бути представлена з допомогою чотирьох станів (рис. 2.13):

1) High Power – високий рівень заряду батареї (наприклад, ≥ 10%);
2) Low Power – критично низький рівень заряду батареї (наприклад, < 10%);
3) Charging – мобільний пристрій заряджається;
4) Off Power – мобільний пристрій повністю розряджений чи вимкнений.

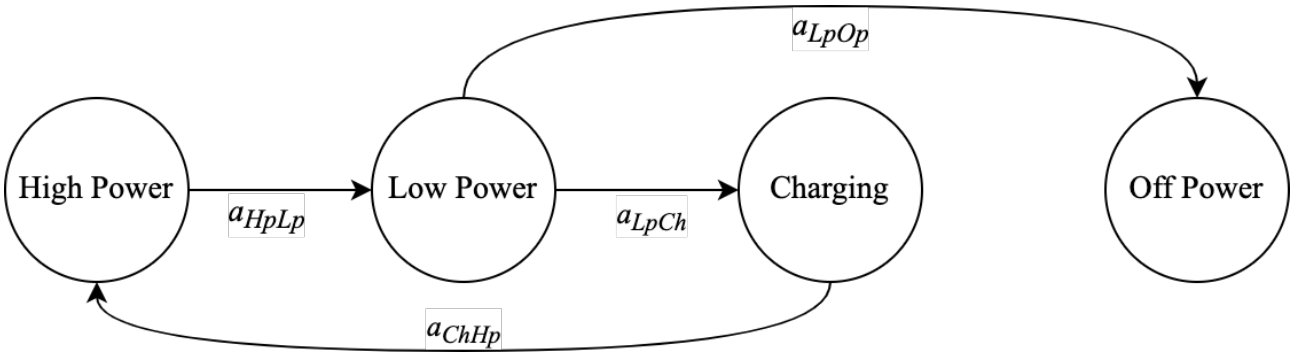


Рис. 2.13. Загальна модель рівня заряду батареї

З точки зору процедури омолодження ПЗ обидва стани «High Power» та «Charging» є такими, в яких її виконання є доцільним. Тому, пропонується спростити дану модель та замінити стани «High Power» та «Charging» на один стан «Stable Power», що показано на рис. 2.14.

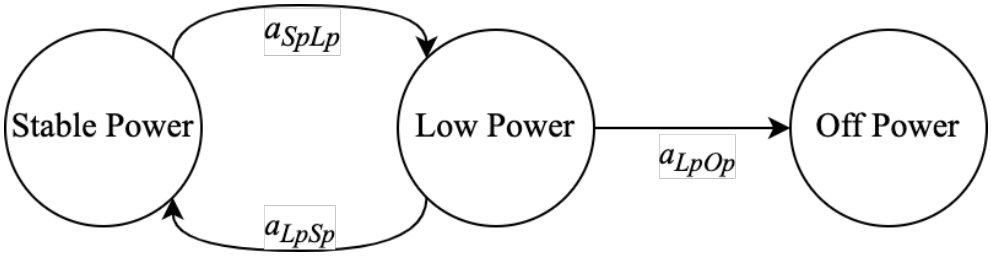


Рис. 2.14. Спрощена модель рівня заряду батареї

Інтенсивності переходів між станами в даній моделі можуть бути обчислені на основі інформації про поточний стан заряду батареї та статистиці попередньої тривалості роботи пристрою до повного розряду батареї. В якості метрик для визначення стану заряду та обчислення інтенсивностей переходів можуть бути використані як відносні значення, що характеризують відсоток заряду, що залишився, так і абсолютні, які характеризують час роботи пристрою з моменту увімкнення та прогнозований час до повного розряду батареї. Зокрема, варто зауважити, що в спрощеній моделі не враховується окремий стан «Charging», однак він має бути

врахований під час обчислення значення інтенсивностей переходів a_{SpLp} та a_{LpSp}, оскільки постійна підзарядка пристрою користувачем може бути вагомим чинником того, що пристрій довго залишатиметься увімкненим.

Графічна модель старіння та омолодження із урахуванням активності використання мобільного пристрою та фактору заряду батареї представлена на рис. 2.15. Важливою особливістю цієї моделі є те, що омолодження ПЗ може виконуватись тільки у стані «Sleep», «Stable Power» та «Young». Такий підхід дозволяє моделювати процес старіння та омолодження, де омолодження не переривається активне використання пристрою користувачем та випереджає перехід батареї у низький стан заряду або повного відключення.

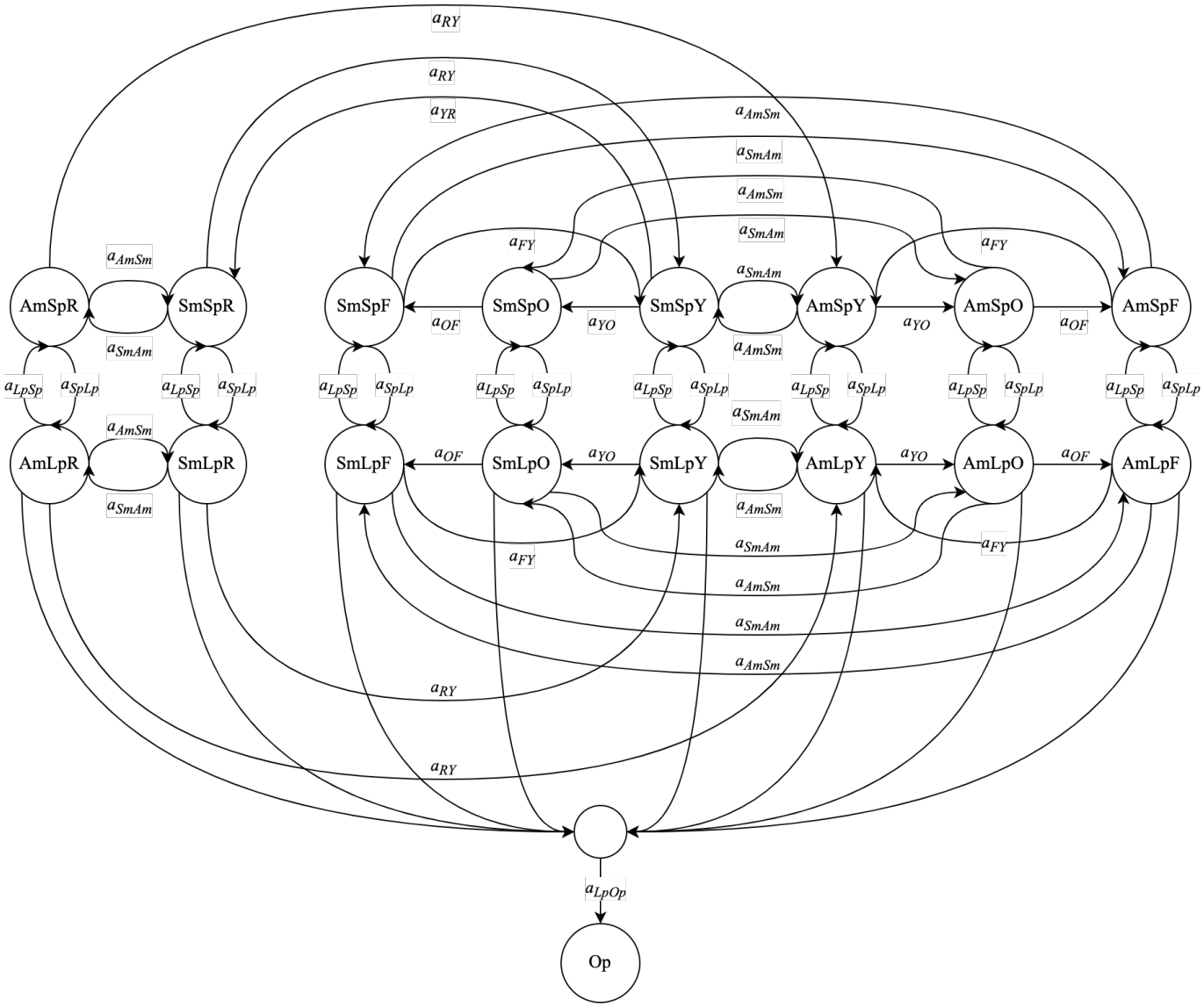


Рис. 2.15. Граф станів та переходів моделі старіння та омолодження ПЗ з урахуванням рівня заряду батареї

Аналітичне представлення моделі у вигляді системи диференційних рівнянь (2.5) на основі СТМС для запропонованої моделі старіння та омолодження з урахуванням рівня заряду батареї:

$$\frac{dP_{SmSpR}(t)}{dt} = -(a_{SmAm} + a_{RY} + a_{SpLp})P_{SmSpR}(t) + a_{SmAm}P_{AmSpR}(t) + a_{OR}P_{SmSpO}(t) + a_{LpSp}P_{SmLpR}(t);$$

$$\frac{dP_{SmSpO}(t)}{dt} = -(a_{SmAm} + a_{OR} + a_{SpLp})P_{SmSpO}(t) + a_{AmSm}P_{AmSpO}(t) + a_{YO}P_{SmSpY}(t) + a_{LpSp}P_{SmLpO}(t);$$

$$\frac{dP_{SmSpY}(t)}{dt} = -(a_{SmAm} + a_{YO} + a_{YF} + \lambda_{SpLp})P_{SmSpY}(t) + a_{AmSm}P_{AmSpY}(t) + a_{SmSpR}P_{RY}(t) + a_{SmSpF}P_{FY}(t) + a_{LpSp}P_{SmLpY}(t);$$

$$\frac{dP_{SmSpF}(t)}{dt} = -(a_{SmAm} + a_{FY} + a_{SpLp})P_{SmSpF}(t) + a_{AmSm}P_{AmSpF}(t) + a_{YF}P_{SmSpY}(t) + a_{LpSp}P_{SmLpF}(t);$$

$$\frac{dP_{AmSpF}(t)}{dt} = -(a_{AmSm} + a_{FY} + a_{SpLp})P_{AmSpF}(t) + a_{SmAm}P_{SmSpF}(t) + a_{LpSp}P_{AmLpF}(t);$$

$$\frac{dP_{AmSpY}(t)}{dt} = -(a_{AmSm} + a_{YO} + a_{SpLp})P_{AmSpY}(t) + a_{SmAm}P_{SmSpY}(t) + a_{FY}P_{AmSpF}(t) + a_{RY}P_{AmSpR}(t) + a_{LpSp}P_{AmLpY}(t);$$

$$\frac{dP_{AmSpO}(t)}{dt} = -(a_{AmSm} + a_{OR} + a_{SpLp})P_{AmSpO}(t) + a_{SmAm}P_{SmSpO}(t) + a_{YO}P_{AmSpY}(t) + a_{LpSp}P_{AmLpO}(t);$$

$$\frac{dP_{AmSpR}(t)}{dt} = -(a_{AmSm} + a_{RY} + a_{SpLp})P_{AmSpR}(t) + a_{SmAm}P_{SmSpR}(t) + a_{OR}P_{AmSpO}(t) + a_{LpSp}P_{AmLpR}(t); \qquad (2.5)$$

$$\frac{dP_{SmLpR}(t)}{dt} = -(a_{SmAm} + a_{RY} + a_{LpSp} + a_{LpOp})P_{SmLpR}(t) + a_{AmSm}P_{AmLpR}(t) + a_{OR}P_{SmLpO}(t) + a_{SpLp}P_{SmSpR}(t);$$

$$\frac{dP_{SmLpO}(t)}{dt} = -(a_{SmAm} + a_{OR} + a_{LpSp} + a_{LpOp})P_{SmLpO}(t) + a_{AmSm}P_{AmLpO}(t) + a_{YO}P_{SmLpY}(t) + a_{SpLp}P_{SmSpO}(t);$$

$$\frac{dP_{SmLpY}(t)}{dt} = -(a_{SmAm} + a_{YO} + a_{LpSp} + a_{LpOp})P_{SmLpY}(t) + a_{AmSm}P_{AmLpY}(t) + a_{SmLpR}P_{RY}(t) + a_{SmLpF}P_{FY}(t) + a_{SpLp}P_{SmSpY}(t);$$

$$\frac{dP_{SmLpF}(t)}{dt} = -(a_{SmAm} + a_{FY} + a_{LpSp} + a_{LpOp})P_{SmLpF}(t) + a_{AmSm}P_{AmLpF}(t) + a_{SpLp}P_{SmSpF}(t);$$

$$\frac{dP_{AmLpF}(t)}{dt} = -(a_{AmSm} + a_{FY} + a_{LpSp} + a_{LpOp})P_{AmLpF}(t) + a_{SmAm}P_{SmLpF}(t) + a_{SpLp}P_{AmSpF}(t);$$

$$\frac{dP_{AmLpY}(t)}{dt} = -(a_{AmSm} + a_{YO} + a_{LpSp} + a_{LpOp})P_{AmLpY}(t) + a_{SmAm}P_{SmLpY}(t) + a_{FY}P_{AmLpF}(t) + a_{RY}P_{AmLpR}(t) + a_{SpLp}P_{AmSpY}(t);$$

$$\frac{dP_{AmLpO}(t)}{dt} = -(a_{AmSm} + a_{OR} + a_{LpSp} + a_{LpOp})P_{AmLpO}(t) + a_{SmAm}P_{SmLpO}(t) + a_{YO}P_{AmLpY}(t) + a_{SpLp}P_{AmSpO}(t);$$

$$\frac{dP_{AmLpR}(t)}{dt} = -(a_{AmSm} + a_{RY} + a_{LpSp} + a_{LpOp})P_{AmLpR}(t) + a_{SmAm}P_{SmLpR}(t) + a_{OR}P_{AmLpO}(t) + a_{SpLp}P_{AmSpR}(t);$$

$$\frac{dP_{Op}(t)}{dt} = a_{LpOp}(P_{SmLpY}(t) + P_{AmLpY}(t) + P_{SmLpO}(t) + P_{AmLpO}(t) + P_{AmLpF}(t) + P_{SmLpF}(t) + P_{AmLpR}(t) + P_{SmLpR}(t)).$$

Обчислюючи систему диференційних рівнянь для різних значень a_{YR}, оптимальний час омолодження може бути визначено за умови, коли ймовірність перебування системи в станах «Active» та «Low Power» є найнижчою, тобто процедура омолодження найменш імовірно буде виконуватись під час використання мобільного пристрою користувачем та випереджатиме розряд батареї.

Для верифікації та аналізу моделі з урахуванням заряду батареї визначено розв'язки для системи диференційних рівнянь (2.5) та системи рівнянь оригінальної моделі [103] з допомогою методу Рунге-Кутти 4-го порядку. Порівняння $P_{AmR}(t) + P_{AmF}(t)$ та $P_{AmLpR}(t) + P_{AmLpF}(t)$ для різних значень T_{YR} $(1/a_{YR})$ обох моделей дозволить проаналізувати вплив фактору заряду батареї на прогноз оптимального часу виконання омолодження.

Інтенсивності переходів між станами є наближеними тестовими значеннями, які підібрано таким чином, щоб мати можливість порівняти результати симуляції моделі старіння та омолодження з урахуванням заряду батареї із результатами експериментальних досліджень моделі без урахування заряду батареї. В цій роботі використовується 1 хвилина як одиниця часу, а активність користувача визначається за весь період однієї доби. Зокрема, припускається, що інтенсивності переходів a_{AS} та a_{SA} є рівні 0.05 (середній час використання мобільного пристрою до переходу в режим очікування є 20 хвилин) та 0.02 (середній час перебування в режимі очікування є 50 хвилин), відповідно. Переходи із стану «Young» в стан «Old», а також із стану «Old» в стан «Failure» тривають по 100 годин, тому $a_{YO} = a_{OF} = 0.00017$. Процедура омолодження та відновлення після відмови старіння вважаються процесами перезавантаження мобільного пристрою, які тривають близько однієї хвилини, тому $a_{FY} = a_{RY} = 1$. У випадку моделі з урахуванням заряду батареї припускається, що час розряду батареї до низького рівня становить 24 години, а потім до повного вимкнення ще одну годину, тобто, $a_{SpLp} = 0.0007$, а $a_{LpOp} = 0.017$. Час, необхідний для відновлення

заряду батареї з низького до стабільного є рівним близько 2 годин, тобто, $a_{LpSp} = 0{,}0083$.

Виконано симуляції процесу старіння та омолодження для різних значень T_{YR} як для оригінальної моделі без урахування заряду батареї (рис. 2.16), так і для моделі з урахуванням нового фактору (рис. 2.17). Оскільки модель старіння та омолодження враховує заряд батареї і активність використання мобільного пристрою, то оптимальним часом виконання омолодження буде тоді, коли ймовірність перебування системи в станах активного використання та низького заряду є найнижчою, тобто $min(P_{AmLpR}(t) + P_{AmLpF}(t))$.

Результати симуляції з допомогою моделі без урахування заряду батареї (рис. 2.16) показують, що оптимальний час омолодження можна вважати рівним $1/a_{YR} = 1620$ хв. (27 год.). Тобто, найменша точка кривої Am + AmF на графіку вказує, що при інтенсивності омолодження $a_{YR} = 0{,}00062$, можна забезпечити найменшу імовірність переривання активного використання системи користувачем процесом омолодження або відмовою старіння.

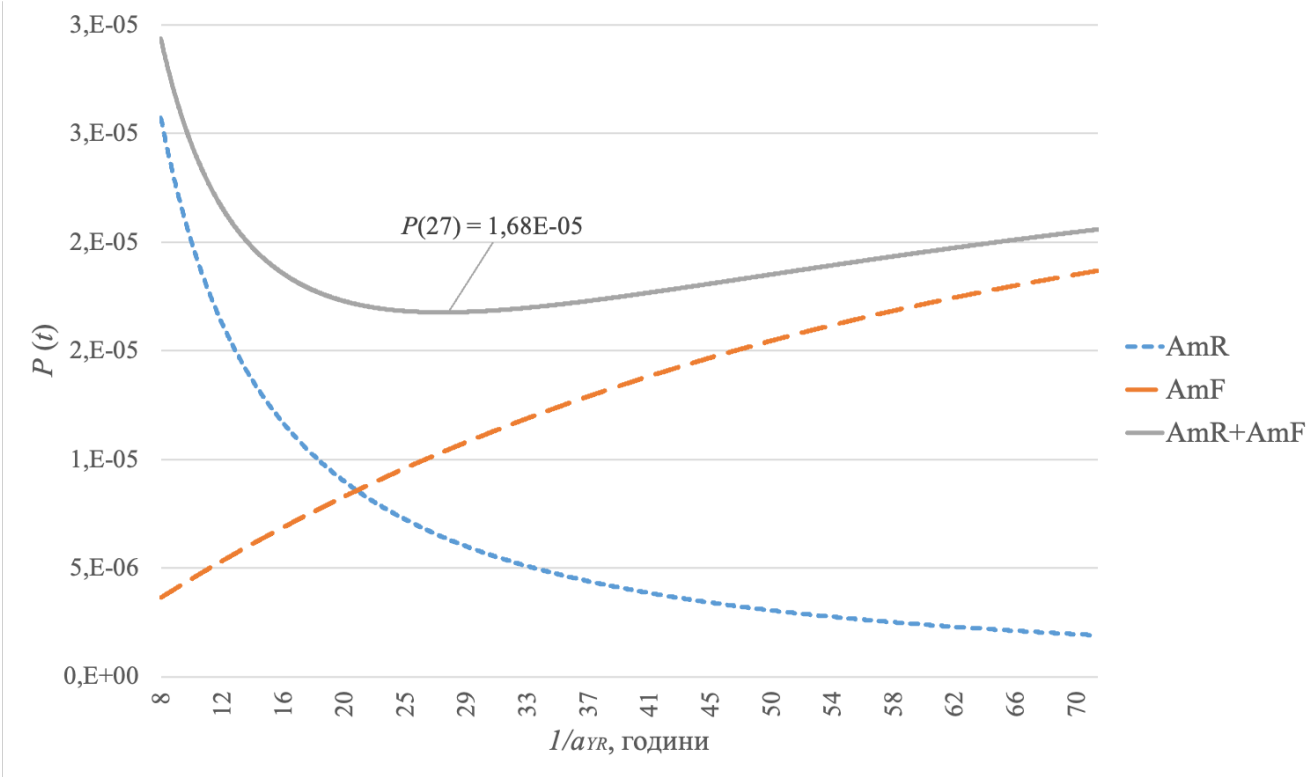


Рис. 2.16. Результати симуляції процесу старіння та омолодження для різних значень T_{YR} з допомогою моделі без урахування заряду батареї

В свою чергу, модель з урахуванням заряду батареї (рис. 2.17) для тих самих умов старіння та активності використання мобільного пристрою показує оптимальний час омолодження рівний близько 360 хвилинам (6 год.), що в 4,5 рази менше, ніж прогноз часу без урахування заряду батареї. В даному випадку за допомогою найменшої точки кривої AmLpR + AmLpF на графіку можна визначити інтенсивності омолодження $a_{YR} = 0{,}0028$. Порівняння отриманих результатів показує, що урахування заряду батареї має значний вплив на результати планування часу виконання омолодження ПЗ.

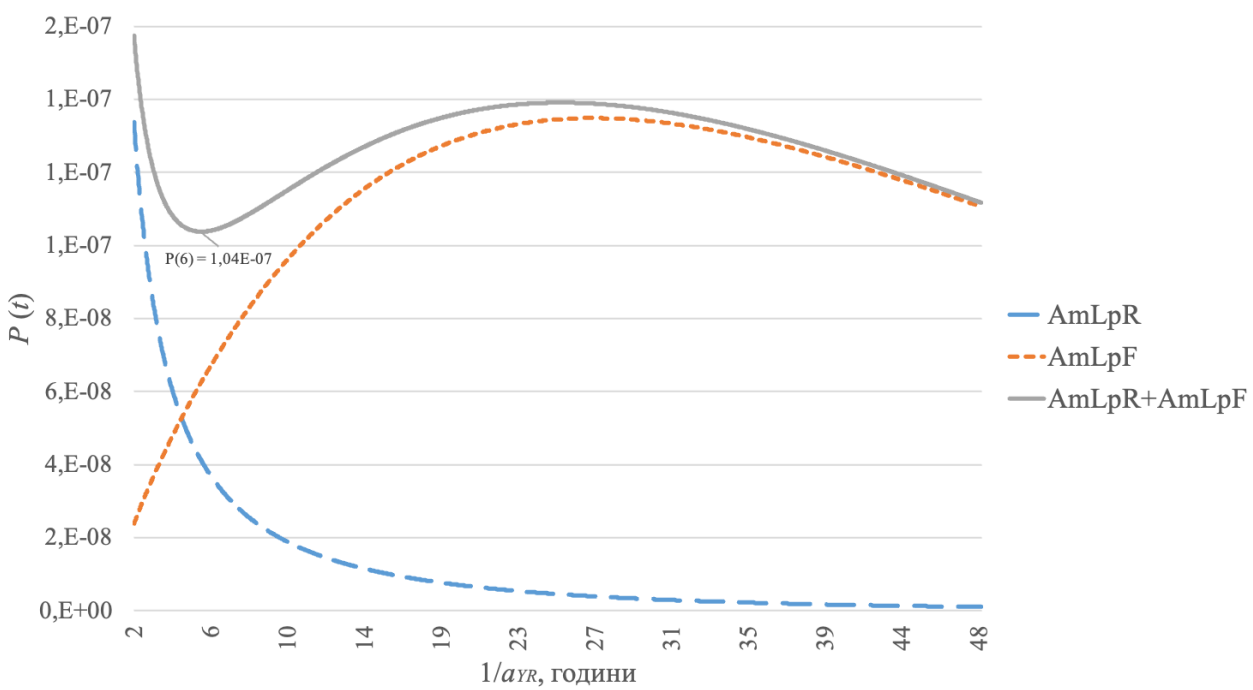


Рис. 2.17. Результати симуляції процесу старіння та омолодження для різних значень T_{YR} з допомогою моделі з урахуванням заряду батареї

Ідея урахування заряду батареї полягає в тому, що виконання процедури омолодження є недоцільним або неможливим в той час, коли мобільний пристрій має критично низький заряд, або повністю розряджений. Отримані результати обчислень підтверджують цю гіпотезу, зокрема рис. 2.18, на якому показано збільшення ймовірності повного розрядження батареї із збільшенням часу виконання процедури омолодження. Якщо ми не враховуємо заряд батареї, то отриманий прогноз оригінальної моделі може не мати практичного результату, оскільки після 27 годин роботи мобільний пристрій буде з великою імовірністю розрядженим. Вплив фактору батареї також спостерігається на рис. 2.17, де $1/a_{YR} > 27$.

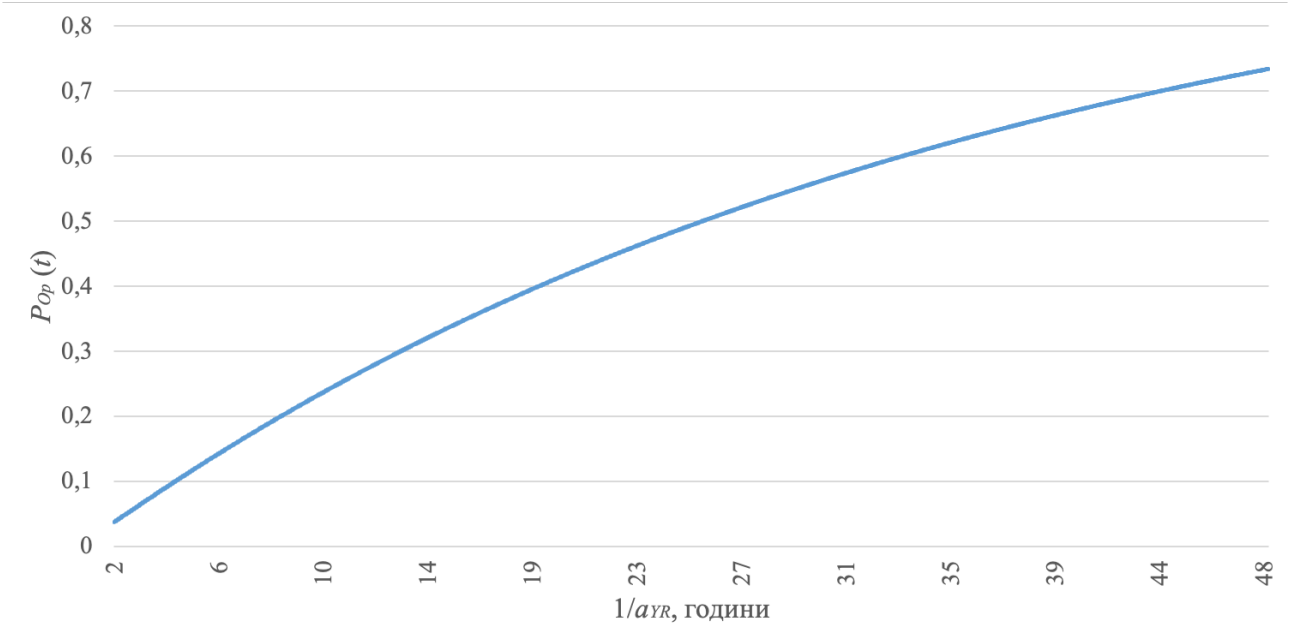


Рис. 2.18. Ймовірність вимкнення мобільного пристрою внаслідок розрядження батареї

Варто зауважити, що запланована процедура омолодження не обов'язкова може призводити до виконання процедури омолодження у вибраний час (наприклад, через 5 год. 50 хв. після увімкнення пристрою), оскільки в цей час система і далі може перебувати в стані «Young», однак батарея вже може бути заряджена. Таким чином, необхідно передбачати додаткові перевірки стану системи та виконувати прогнозування наступного часу виконання омолодження.

Обчислені значення ймовірностей перебування в станах «Failure» та «Rejuvenation» знаходяться в межах 10^{-7}, що потребує пояснення. В даній роботі припускається, що ці два стани, на відміну від інший можливих станів системи, тривають тільки час, який необхідний для виконання процедури омолодження, наприклад, для виконання перезавантаження системи. Таким чином, висока ймовірність перебування в цих станах буде означати те, що система постійно перебуває в непрацездатному стані простою, що є неприйнятним для користувача мобільного пристрою.

2.4. Комплексна модель старіння та омолодження для операційної системи Android

В попередніх підрозділах було запропоновано окремі моделі, які враховують різні фактори, що прямо чи опосередковано впливають на процес старіння. Для

розроблення методів омолодження ПЗ важливо сформулювати та описати комплексну модель, яка враховує всі переваги попередніх двох моделей. В даній роботі пропонується комплексна модель [6] (рис. 2.19) старіння та омолодження на основі ланцюгів Маркова з неперервним часом розподілу, яка враховує різні рівні старіння [5], активність використання мобільного пристрою користувачем [103], а також стан заряду батареї [7]. Модель описує систему, яка може перебувати в таких станах:

- Active Mobile + Stable Power + Young (AmSpY) – користувач активно використовує мобільний пристрій з високим рівнем заряду і продуктивності;
- Active Mobile + Stable Power + Aging (AmSpA) – користувач активно використовує мобільний пристрій з високим рівнем заряду, де спостерігається погіршення продуктивності, але відмови старіння малоймовірні;
- Active Mobile + Stable Power + Old (AmSpO) – користувач активно використовує мобільний пристрій з високим рівнем заряду, де спостерігається низький рівень продуктивності, що з високою ймовірністю може призводити до відмов старіння;
- Active Mobile + Stable Power + Failure (AmSpF) – стан відмови старіння під час активного використання мобільного пристрою з високим зарядом батареї, користувач змушений перезавантажити мобільний пристрій;
- Active Mobile + Stable Power + Rejuvenation (AmSpR) – виконується процедура омолодження під час активного використання користувачем мобільного пристрою з високим зарядом батареї;
- Sleep Mobile + Stable Power + Young (SmSpY) – мобільний пристрій в режимі очікування з високим зарядом і продуктивністю;
- Sleep Mobile + Stable Power + Aging (SmSpA) – мобільний пристрій в режимі очікування з високим зарядом, де спостерігається погіршення продуктивності, але відмови старіння малоймовірні;
- Sleep Mobile + Stable Power + Old (SmSpO) – мобільний пристрій в режимі очікування з високим зарядом, де спостерігається низький рівень продуктивності, що з високою ймовірністю може призводити до виникнення відмов старіння;

- Sleep Mobile + Stable Power + Failure (SmSpF) – стан відмови старіння в режимі очікування мобільного пристрою з високим зарядом батареї, система самостійно перезавантажує мобільний пристрій;

- Sleep Mobile + Stable Power + Rejuvenation (SmSpR) – виконується процедура омолодження в режимі очікування мобільного пристрою з високим зарядом батареї;

- Active Mobile + Low Power + Young (AmLpY) – користувач активно використовує мобільний пристрій з низьким рівнем заряду і високою продуктивністю;

- Active Mobile + Low Power + Aging (AmLpA) – користувач активно використовує мобільний пристрій з низьким рівнем заряду, де спостерігається погіршення продуктивності, але відмови старіння малоймовірні;

- Active Mobile + Low Power + Old (AmLpO) – користувач активно використовує мобільний пристрій з низьким рівнем заряду, де спостерігається низький рівень продуктивності, що з високою ймовірністю може призводити до відмов старіння;

- Active Mobile + Low Power + Failure (AmLpF) – стан відмови старіння під час активного використання мобільного пристрою з низьким зарядом батареї, користувач змушений перезавантажити мобільний пристрій;

- Active Mobile + Low Power + Rejuvenation (AmLpR) – виконується процедура омолодження під час активного використання користувачем мобільного пристрою з низьким зарядом батареї;

- Sleep Mobile + Low Power + Young (SmLpY) – мобільний пристрій в режимі очікування з низьким зарядом і високою продуктивністю роботи;

- Sleep Mobile + Low Power + Aging (SmLpA) – мобільний пристрій в режимі очікування з низьким зарядом, де спостерігається погіршення продуктивності, але відмови старіння малоймовірні;

- Sleep Mobile + Low Power + Old (SmLpO) – мобільний пристрій в режимі очікування з низьким зарядом, де спостерігається низький рівень продуктивності, що з високою ймовірністю може призводити до виникнення відмов старіння;

- Sleep Mobile + Low Power + Failure (SmLpF) – стан відмови старіння в режимі очікування мобільного пристрою з низьким зарядом батареї, система самостійно перезавантажує мобільний пристрій;
- Sleep Mobile + Low Power + Rejuvenation (SmLpR) – виконується процедура омолодження в режимі очікування мобільного пристрою з низьким зарядом батареї;
- Off Power (Op) – стан повного розрядження батареї.

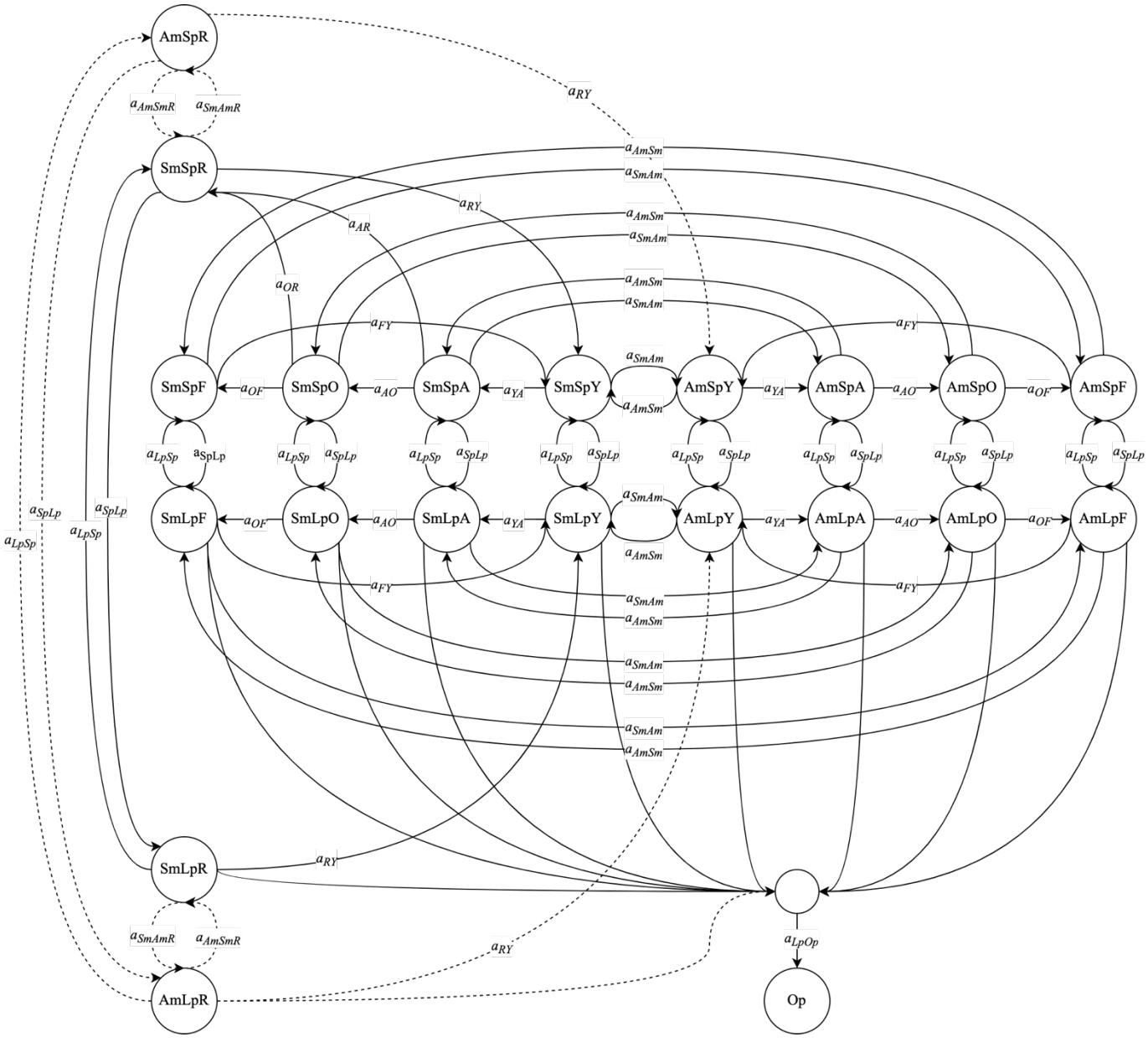


Рис. 2.19. Граф станів та переходів комплексної моделі старіння та омолодження ПЗ для ОС Android

Аналітичне представлення моделі у вигляді системи диференційних рівнянь (2.6) на основі СТМС комплексної моделі старіння та омолодження:

$$\frac{dP_{AmSpY}(t)}{dt} = -(a_{AmSm} + a_{SpLp} + a_{YA})P_{AmSpY}(t) + a_{SmAm}P_{SmSpY}(t) + a_{LpSp}P_{AmLpY}(t) + \\ + a_{FY}P_{AmSpF}(t) + a_{RY}P_{AmSpR}(t);$$

$$\frac{dP_{AmSpA}(t)}{dt} = -(a_{AmSm} + a_{SpLp} + a_{AO})P_{AmSpA}(t) + a_{SmAm}P_{SmSpA}(t) + a_{LpSp}P_{AmLpA}(t) + a_{YA}P_{AmSpY}(t);$$

$$\frac{dP_{AmSpO}(t)}{dt} = -(a_{AmSm} + a_{SpLp} + a_{OF})P_{AmSpO}(t) + a_{SmAm}P_{SmSpO}(t) + a_{LpSp}P_{AmLpO}(t) + a_{AO}P_{AmSpA}(t);$$

$$\frac{dP_{AmSpF}(t)}{dt} = -(a_{AmSm} + a_{SpLp} + a_{FY})P_{AmSpF}(t) + a_{SmAm}P_{SmSpF}(t) + a_{LpSp}P_{AmLpF}(t) + a_{OF}P_{AmSpO}(t);$$

$$\frac{dP_{AmLpY}(t)}{dt} = -(a_{AmSm} + a_{LpSp} + a_{YA} + a_{LpOp})P_{AmLpY}(t) + a_{SmAm}P_{SmLpF}(t) + a_{SpLp}P_{AmSpY}(t) + \\ + a_{FY}P_{AmLpF}(t) + a_{RY}P_{AmLpR}(t);$$

$$\frac{dP_{AmLpA}(t)}{dt} = -(a_{AmSm} + a_{LpSp} + a_{AO} + a_{LpOp})P_{AmLpA}(t) + a_{SmAm}P_{SmLpA}(t) + a_{SpLp}P_{AmSpA}(t) + \\ + a_{YA}P_{AmLpY}(t);$$

$$\frac{dP_{AmLpO}(t)}{dt} = -(a_{AmSm} + a_{LpSp} + a_{OF} + a_{LpOp})P_{AmLpO}(t) + a_{SmAm}P_{SmLpO}(t) + a_{SpLp}P_{AmSpO}(t) + \\ + a_{AO}P_{AmLpA}(t);$$

$$\frac{dP_{AmLpF}(t)}{dt} = -(a_{AmSm} + a_{LpSp} + a_{FY} + a_{LpOp})P_{AmLpF}(t) + a_{SmAm}P_{SmLpF}(t) + a_{SpLp}P_{AmSpF}(t) + \\ + a_{OF}P_{AmLpO}(t);$$

$$\frac{dP_{SmSpY}(t)}{dt} = -(a_{SmAm} + a_{SpLp} + a_{YA})P_{SmSpY}(t) + a_{AmSm}P_{AmSpY}(t) + a_{LpSp}P_{SmLpY}(t) + \\ + a_{FY}P_{SmSpF}(t) + a_{RY}P_{SmSpR}(t);$$

$$\frac{dP_{SmSpA}(t)}{dt} = -(a_{SmAm} + a_{SpLp} + a_{AO} + a_{AR})P_{SmSpA}(t) + a_{AmSm}P_{AmSpA}(t) + a_{LpSp}P_{SmLpA}(t) + \\ + a_{YA}P_{SmSpY}(t);$$

$$\frac{dP_{SmSpO}(t)}{dt} = -(a_{SmAm} + a_{SpLp} + a_{OF} + a_{OR})P_{SmSpO}(t) + a_{AmSm}P_{AmSpO}(t) + a_{LpSp}P_{SmLpO}(t) + \\ + a_{AO}P_{SmSpA}(t);$$

(2.6)

$$\frac{dP_{SmSpF}(t)}{dt} = -(a_{SmAm} + a_{SpLp} + a_{FY})P_{SmSpF}(t) + a_{AmSm}P_{AmSpF}(t) + a_{LpSp}P_{SmLpF}(t) + a_{OF}P_{SmSpO}(t);$$

$$\frac{dP_{SmLpY}(t)}{dt} = -(a_{SmAm} + a_{LpSp} + a_{YA} + a_{LpOp})P_{SmLpY}(t) + a_{AmSm}P_{AmLpY}(t) + a_{SpLp}P_{SmSpY}(t) + \\ + a_{FY}P_{SmLpF}(t) + a_{RY}P_{SmLpR}(t);$$

$$\frac{dP_{SmLpA}(t)}{dt} = -(a_{SmAm} + a_{LpSp} + a_{AO} + a_{LpOp})P_{SmLpA}(t) + a_{AmSm}P_{AmLpA}(t) + a_{SpLp}P_{SmSpA}(t) + \\ + a_{YA}P_{SmLpY}(t);$$

$$\frac{dP_{SmLpO}(t)}{dt} = -(a_{SmAm} + a_{LpSp} + a_{OF} + a_{LpOp})P_{SmLpO}(t) + a_{AmSm}P_{AmLpO}(t) + a_{SpLp}P_{SmSpO}(t) + \\ + a_{AO}P_{SmLpA}(t);$$

$$\frac{dP_{SmLpF}(t)}{dt} = -(a_{SmAm} + a_{LpSp} + a_{FY} + a_{LpOp})P_{SmLpF}(t) + a_{AmSm}P_{AmLpF}(t) + a_{SpLp}P_{SmSpF}(t) + \\ + a_{OF}P_{SmLpO}(t);$$

$$\frac{dP_{SmSpR}(t)}{dt} = -(a_{SmAmR} + a_{SpLp} + a_{RY})P_{SmSpR}(t) + a_{AmSmR}P_{AmSpR}(t) + a_{LpSp}P_{SmLpR}(t) + \\ + a_{OR}P_{SmSpO}(t) + a_{AR}P_{SmSpA}(t);$$

$$\frac{dP_{AmSpR}(t)}{dt} = -(a_{AmSmR} + a_{SpLp} + a_{RY})P_{AmSpR}(t) + a_{SmAmR}P_{SmSpR}(t) + a_{LpSp}P_{AmLpR}(t);$$

$$\frac{dP_{SmLpR}(t)}{dt} = -(a_{SmAmR} + a_{LpSp} + a_{RY} + a_{LpOp})P_{SmLpR}(t) + a_{AmSmR}P_{AmLpR}(t) + a_{SpLp}P_{SmSpR}(t);$$

$$\frac{dP_{AmLpR}(t)}{dt} = -(a_{AmSmR} + a_{LpSp} + a_{RY} + a_{LpOp})P_{AmLpR}(t) + a_{SmAmR}P_{SmLpR}(t) + a_{SpLp}P_{AmSpR}(t);$$

$$\frac{dP_{Op}(t)}{dt} = a_{LpOp}(P_{AmLpR}(t) + P_{SmLpR}(t) + P_{SmLpF}(t) + P_{SmLpO}(t) + P_{SmLpA}(t) + P_{SmLpA}(t) + \\ + P_{SmLpY}(t) + P_{AmLpY}(t) + P_{AmLpA}(t) + P_{AmLpO}(t) + P_{AmLpF}(t)).$$

Враховуючи попередні результати досліджень та характеристики можливих станів системи, виконання процедури омолодження буде доцільним при умові, коли система перебуває у стані «Sleep», «Stable Power» та «Aging». Тобто, процедура омолодження в стані SmSpA не буде мати значного негативного впливу на UX, а також випереджатиме перехід пристрою у вразливий до відмов старіння стан чи до розряду батареї. Виконання процедури омолодження в комплексній моделі описано

переходом зі стану SmSpA в стан SmSpR з інтенсивністю a_{AR} (середнім часом T_{AR}). З цього випливає, що прогнозування оптимального часу омолодження полягає у пошуку такого значення a_{AR} (T_{AR}), при якому система з найбільшою імовірністю перебуває в стані SmSpA в момент виконання процедури. Для пошуку оптимального часу виконання омолодження необхідно забезпечити наступні умови:

$$\begin{cases} \sum\limits_{i \in S_A} P_i(t) \to \max; \\ \sum\limits_{j \in S_{FR}} P_j(t) \to \min; \\ P_{SmSpA}(t) \to \max; \end{cases} \qquad (2.7)$$

де S_A – це множина працездатних станів системи (AmSpY, AmSpA, AmSpO, SmSpY, SmSpA, SmSpO, AmLpY, AmLpA, AmLpO, SmLpY, SmLpA, SmLpO), S_{FR} – це множина непрацездатних станів системи (AmSpF, AmSpR, SmSpF, SmSpR, AmLpF, AmLpR, SmLpF, SmLpR, Op).

2.5. Висновки до розділу 2

У даному розділі для математичного опису процесу старіння та омолодження ПЗ використано апарат СТМС, який дозволив побудувати як графічні моделі у вигляді графів станів та переходів, так і аналітичні у вигляді систем диференційних рівнянь Колмогорова-Чепмена.

Виконано аналіз існуючої моделі процесу старіння та омолодження ПЗ для ОС Android з урахуванням активності використання мобільного пристрою користувачем та визначено основні недоліки цієї моделі: омолодження може бути недоцільним і надлишковим у зв’язку із його виконанням в стані «Young», коли перехід в стан «Old» не спостерігатиметься; виконання омолодження в стані активного використання мобільного пристрою є неможливим, оскільки воно полягає у перезавантаженні ОС чи важливих для роботи компонент; не передбачено механізми «теплого» та «холодного» омолодження, які б передбачали різні стратегії виконання омолодження та забезпечували можливість одночасного перебування в стані омолодження і активного використання.

Запропоновано моделі старіння та омолодження ПЗ для ОС Android, які враховують як активність використання мобільного пристрою користувачем, так і

механізми «теплого» та «холодного» омолодження, різні рівні старіння та можливі стратегії виконання процедури омолодження, а також, фактор рівня заряду батареї.

Симуляції процесу старіння та омолодження з допомогою моделі з урахуванням різних рівнів старіння ПЗ показали, що ефективною стратегією омолодження ПЗ є повторюване виконання процедури омолодження в стані «Aging», яке дозволяє збільшити $P_{AmY}(t)$ в 1,37-1,77 рази порівняно із симуляцією без виконання омолодження. Запропонована модель дозволяє виконувати омолодження тільки при необхідності в станах «Aging» та «Old», тобто тоді, коли в системі спостерігаються процеси старіння, а не в стані «Young», що може покращити як характеристики надійності, так і характеристики UX. Практичне значення моделі полягає у можливості застосування методів на основі вимірювань та порогів метрик старіння для визначення стану системи та формування статистики.

Запропонована модель описує процес старіння з урахуванням «теплого» та «холодного» механізму омолодження. Хоча у виконаній симуляції різниця ймовірності перебування в стані AmY для «теплого» та «холодного» омолодження є незначною і складає в середньому 0,157%, однак значення різниці може залежати від реальних умов старіння та омолодження, тому урахування механізму омолодження є важливим завданням під час прогнозування часу омолодження в реальній системі.

Моделювання процесу старіння та омолодження з урахуванням фактору заряду батареї показало, що рівень заряду може мати вагомий вплив на результат прогнозу часу виконання омолодження ПЗ. Зокрема, виконані симуляції підтверджують, що модель з урахуванням заряду батареї дозволяє отримати прогнозований час виконання омолодження в 4,5 рази менший, ніж аналогічний прогноз без урахування того факту, що батарея розрядиться швидше, ніж відбудеться омолодження.

Для запропонованої комплексної моделі старіння та омолодження визначено умови пошуку оптимального часу виконання омолодження з урахуванням наборів працездатних та непрацездатних станів системи, де непрацездатні стани включають як стани відмови, так і стани простою під час виконання омолодження.

Розроблена в даній роботі комплексна модель старіння та омолодження є ефективним інструментом для проектування та вибору параметрів методу

омолодження ПЗ, що дає змогу формувати технічне завдання на його розроблення. Запропоновані моделі старіння та омолодження можуть бути застосовані для оцінки показників якості існуючого ПЗ. Комплексна модель дозволяє отримувати вирази для показників ефективності омолодження ПЗ.

РОЗДІЛ 3. МЕТОД ОМОЛОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ ANDROID

3.1. Метрики старіння програмного забезпечення в операційній системі Android та їх ефективність

Важливим завданням дослідження старіння ПЗ є виявлення ефективних метрик старіння. Зокрема, для використання моделей, описаних в попередньому розділі, важливо дослідити метрики, які б дозволили достовірно визначати поточний рівень старіння в системі з точки зору продуктивності GUI, а також, визначати активність використання мобільного пристрою користувачем. Для ОС Android в різних дослідженнях [89, 91] використовуються наступні метрики, які описані в таблиці 3.1.

Таблиця 3.1

Метрики старіння ПЗ в ОС Android

Категорія	**Назва**	**Опис**
GUI	Activity Launch Time, або ALT	Тривалість процесу запуску екранів застосунку від моменту ініціалізації до відображення на дисплеї.
Оперативна пам’ять	Free Total / Cached / Cached PSS; Used Total / PSS / Buffers / Shared Memory / Slab; Lost RAM; ZRAM Physically Used / In Swap; KSM Saved / Shared / Unshared / Volatile.	Глобальні метрики пам’яті та використання пам’яті процесами.
Файлове сховище	Reads Completed / Merged; Writes Completed / Merged; Sectors Read / Written; Reading / Writing Time;	Метрики читання і запису.

Категорія	**Назва**	**Опис**
	Read / Write Completion Time; I/O Time; Weighted I/O Time.	
Збирач сміття	Concurrent GC; Explicit GC; GC Pause Time; GC Duration Time.	Кількість подій паралельного та явного збирача сміття, тривалість затримок та виконання збирача сміття.
Процеси та CPU	Minor / Major Page Fault; User / System Time; CPU / Wait Time.	Кількість помилок сторінки, час роботи процесів у різних режимах, час роботи та очікування CPU.

Результати аналізу старіння ПЗ [91] показали, що робоче навантаження, технічні характеристики пристрою і версія ОС впливають на продуктивність, а тренди в погіршенні продуктивності корелюються з використанням пам’яті процесами і роботою збирача сміття. В роботі [21] визначено групу метрик для визначення проявів старіння в ОС Android, що мають високу статистичну достовірність, зокрема, це метрики System Server PSS, ALT, GC Pause Time та GC Duration Time.

Важливими метриками використання пам’яті є Total Used, Total Free та PSS окремих процесів.

PSS – це частина оперативної пам’яті, що зайнята певним процесом і складається з сторінок приватної пам’яті цього процесу та частини сторінок спільної пам’яті одного чи декількох процесів (наприклад, якщо три процеси мають 3 mb спільної пам’яті, то на кожен процес припадає 1 mb пам’яті PSS). В свою чергу, RAM розділено на сторінки, кожна із яких зазвичай займає 4 kb пам’яті.

System Server – це процес, що працює протягом усього періоду служби ОС Android, ініціалізує шар Application Framework і запускає майже всі системні сервіси

Java. Експерименти показали, що процес System Server зазнає значного збільшення PSS.

ALT – це кількість часу, що пройшло від моменту ініціалізації запуску Android Activity до його остаточного відображення на дисплеї мобільного пристрою. Android Activity – це основний елемент ОС Android представлений однойменним базовим класом Android SDK, який забезпечує можливість реалізації UI та взаємодії користувача із застосунком. У більшості випадків Android Activity – це повноекранне вікно застосунку, хоча в режимі плаваючого вікна може займати тільки частину фізичного дисплею.

Різні автори [21, 25] використовували ALT в якості метрики для визначення наявності старіння ПЗ в контексті продуктивності UI, що має безпосередній вплив на UX. В [91] застосунки, які піддавались робочому навантаженню, перезавантажувались кожні 60 секунд для того, щоб забезпечити регулярний збір даних про тривалість запуску Android Activity протягом всього часу виконання експерименту, а також, щоб уникнути кешування Activity системою Android і запобігти накопиченню помилок (таких як витоків пам'яті) всередині користувацьких застосунків, оскільки в дослідженні акцентували увагу на старінні ПЗ в середині самої ОС Android. Тобто, такий підхід із перезавантаженням не враховує старіння користувацьких застосунків та можливості перевірки реальних сценаріїв використання мобільного пристрою, де перезавантаження не передбачені.

Ефективні метрики продуктивності GUI мають забезпечувати достатньою кількістю даних для різних варіантів використання. Android застосування зазвичай мають тільки одну головну Activity, тому частота запусків Activity під час реального використання мобільного пристрою може бути низькою. Зокрема, у випадку крос-платформових фреймворків та застосунків майже вся логіка та користувацький інтерфейс реалізується в межах одного Android Activity екрану, який слугує контейнером крос-платформового застосунку і забезпечує інтерфейс взаємодії крос-платформового SDK із ОС Android. Слід також враховувати різні випадки використання програм користувачами. Наприклад, коли користувач може витрачати значну кількість часу і регулярно використовувати тільки один застосунок, не

переходячи на інший. Тому, використання метрики ALT, як і необхідність в перезапуску застосунків для збору інформації під час виконання тестів, накладають певні обмеження на можливість коректної оцінки наявності старіння в системі в реальних варіантах використання мобільного пристрою з урахуванням поведінки користувача.

В даній роботі пропонується розглянути нові метрики UI [3, 9] в контексті старіння ПЗ, що були б застосовними для різних сценаріїв використання та могли б бути виміряними протягом всього життєвого циклу користувацьких застосунків. Відображення UI полягає у послідовному оновленні окремих кадрів на дисплеї мобільного пристрою, які будуються на основі інформації про активні користувацькі застосунки, що знаходяться на передньому плані. Плавна робота застосунків характеризується оптимальною кількістю кадрів у секунду, яка має бути в межах 60 FPS. Крім того, перевантаження обчисленнями основного потоку відповідального за UI призводить до затримок у відображенні кадрів, що в свою чергу спричиняє втрату частини наступних кадрів. Тому для забезпечення плавності роботи застосунків важливо мінімізувати кількість пропущених чи затриманих, або, так званих, «зіпсованих» кадрів. Таким чином, метриками, які б задовольнили описані вище умови та вимоги в контексті виявлення старіння, є тривалість відображення кадрів графічного інтерфейсу користувача та кількість «зіпсованих» кадрів під час відображення користувацьких застосунків.

FDT – це різниця між `FRAME_COMPLETED` і `INTENDED_VSYNC`, які можна отримати з допомогою команди `gfxinfo` утиліти командного рядка Android `dumpsys` [107]. Необхідно, щоб ця тривалість відображення кадру не перевищувала 16 мс, щоб забезпечити частоту оновлення кадрів понад 60 кадрів у секунду.

JFC – це пропущені чи затримані кадри, які не були відображені. Відношення JFC до загальної кількості кадрів, які мали бути відображені, є часткою пропущених чи затриманих кадрів (JFR).

Для обґрунтування доцільності використання нових метрик та необхідності їх подальших досліджень в контексті старіння мобільного ПЗ виконано ряд науково-прикладних задач, зокрема, розроблено фреймворк для симуляції старіння в ОС

Android шляхом виконання стресових тестів, який також дає змогу вимірювати нововведені метрики та виконувати аналіз їх ефективності у порівнянні із іншими метриками старіння на рівні системи та застосунків. Детальний опис розробленого фреймворку подано в розділі 4.2.

Для аналізу ефективності запропонованих метрик старіння ПЗ виконано два експериментальні дослідження [3, 4, 12], які враховують різні умови та фактори старіння. Детальна конфігурація досліджень представлена у додатку А.

Метою виконання першого експерименту (див. EXP1 у додатку Б) було порівняння метрик відображення кадрів інтерфейсу із метрикою ALT, оцінка можливості застосування нових метрик для визначення наявності старіння в системі та прогнозування часу до відмови внаслідок старіння, а також, виявлення взаємозв'язків метрик відображення кадрів із метриками пам'яті та збирача сміття. Застосовуючи метод лінійної регресії для метрик ALT, FDT та JFC можна робити висновки про зміни в продуктивності системи та виявляти тренди старіння ПЗ. Припускається, що FDT чи JFC можуть розглядатися в якості метрик старіння, якщо їх знак коефіцієнта лінійної регресії співпадає із знаком коефіцієнта лінійної регресії ALT. Але також припускається, що метрики кадрів можуть бути більш репрезентативні і такі, що показують тренд старіння в тих випадках, коли метрика ALT не може бути використана. Для визначення кореляцій між окремими індикаторами, зокрема для визначення впливу використання пам'яті на метрики кадрів, застосовується коефіцієнт кореляції Спірмена.

Наступне експериментальне дослідження (див. EXP2 у додатку Б) старіння ПЗ в ОС Android дозволило детальніше обґрунтувати доцільність застосування метрик відображення кадрів для виявлення старіння, зокрема, в різних сценаріях використання мобільного пристрою користувачем. У виконаному експерименті враховувались два сценарії, один з яких передбачає постійне генерування робочого навантаження, а інший генерує робоче навантаження протягом 30 хвилин з 20-ти хвилинними паузами. Виявлення наявності чи відсутності старіння в тестовому випадку відбувалось з допомогою аналізу тренду часового ряду непараметричним методом Манна-Кендала, а криву нахилу тренду визначено з використанням

процедури Сена. Рівень значущості для отриманих результатів має бути більшим за 90% ($\alpha = 0.1$). Тобто, якщо значення $P < 0.1$, а нахил тренду більший нуля, то спостерігається тренд старіння для відповідної метрики у відповідному тестовому випадку.

Примусовий перезапуск застосунків кожні 30 секунд в тестових випадках першого експерименту (рис. 3.1) дозволяє отримувати регулярні записи метрики ALT так само, як і FDT. Однак, такий підхід до виконання експерименту унеможливлює перевірку реальних випадків використання користувацьких застосунків, наприклад, коли користувач постійно працює із одним застосунком, чи цей застосунок реалізовано з допомогою тільки однієї Android Activity.

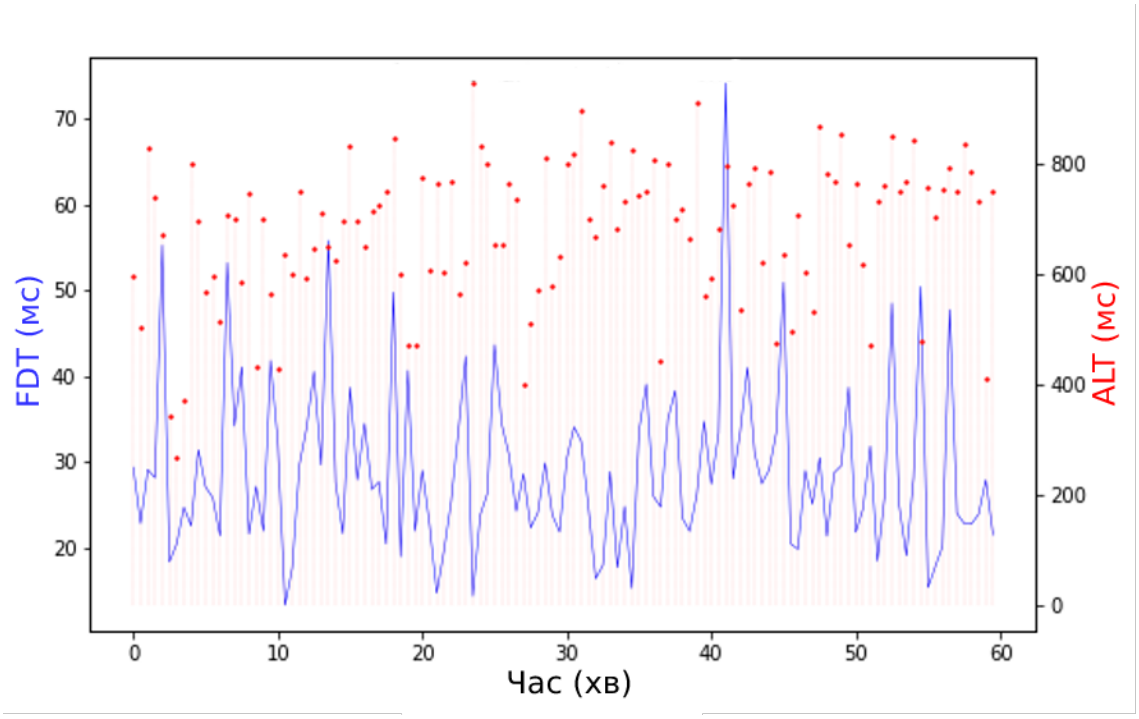


Рис. 3.1. Порівняння метрик ALT та FDT для тестового випадку із примусовим перезапуском застосунків кожні 30 секунд

Результати виконання тестових випадків без примусового перезавантаження застосунків в першому експерименті (рис. 3.2) показують, що метрика FDT може надати більш деталізовану картину процесу старіння. Між кожним стовпцем, який описує значення метрики ALT, метрика FDT дозволяє оцінити продуктивність UI вже під час використання відкритого користувацького застосунку. Метрика ALT в тестах без перезапусків показує приблизно в чотири рази менше записів, ніж метрика

тривалості відображення кадру. Крім того, варто відзначити, що у виконаних експериментах відбувається запуск відразу п’ятьох різних користувацьких застосунків, що впливає на кількість значень метрики ALT. В свою чергу, значення метрики FDT можуть залежати від складності UI та інтенсивності використання мобільного пристрою користувачем, що буде перевірено у наступному експерименті. Отже, експериментальним шляхом підтверджується гіпотеза про те, що технічні особливості метрики ALT обмежують її використання в контексті виявлення старіння ПЗ, а метрика FDT є більш репрезентативною та практично застосовуваною.

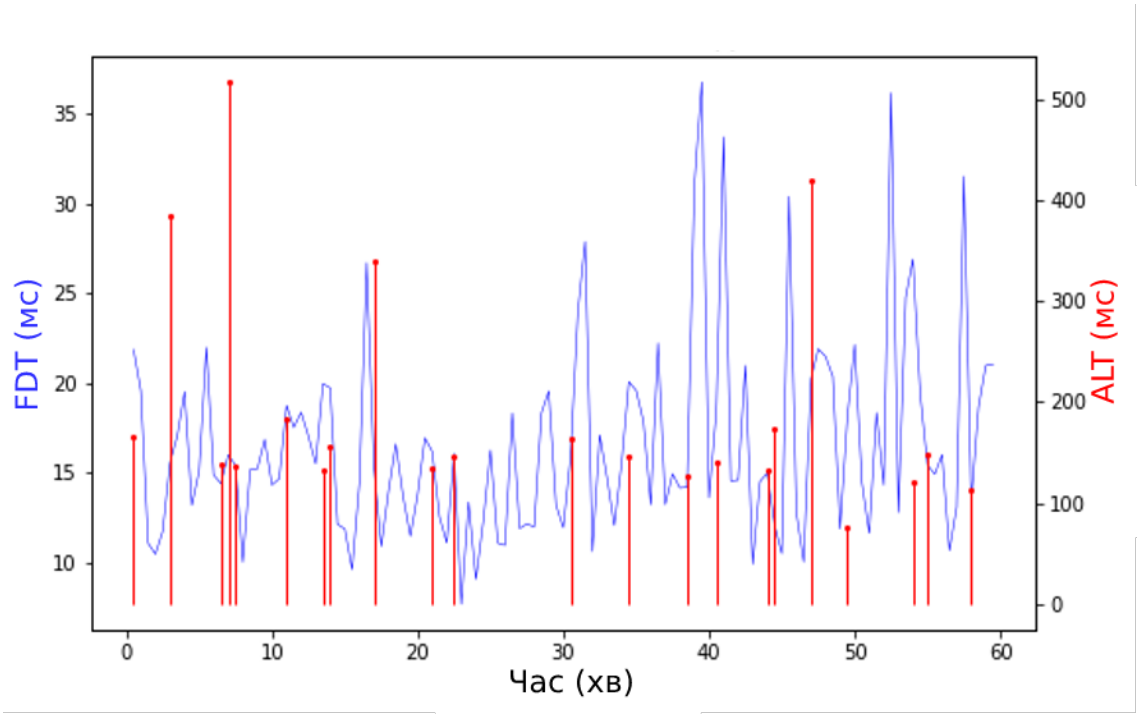


Рис. 3.2. Порівняння записів метрик ALT та FDT без примусового перезапуску застосунків

Для виявлення трендів старіння ПЗ застосовано метод лінійної регресії, який використовує часові ряди відповідних метрик. Застосовано метод лінійної регресії (рис. 3.3) для метрики FDT у другому експерименті з перезапуском застосунків. Результати експерименту показують, що тривалість відображення кадрів вже перевищує порогове значення 16 мс, яке б забезпечувало 60 кадрів в секунду. Враховуючи наявність тренду до збільшення тривалості відображення кадру, можна було б застосувати методи прогнозування для визначення часу, коли тривалість відображення кадрів буде критично високою і зможе спричинити відмову старіння.

Тобто, метрика FDT може бути застосована для визначення стану старіння системи та для прогнозування TTAF. Відмовою через старіння в цьому випадку можна вважати той стан системи, коли тривалість відображення кадру значно перевищує 16 мс, наприклад, понад 48 мс. Таким чином, можна виділити три стани системи в контексті старіння: система в молодому стані (до 16 мс); система в стані переходу до старіння (16-48 мс); система в стані відмови через старіння (понад 48 мс).

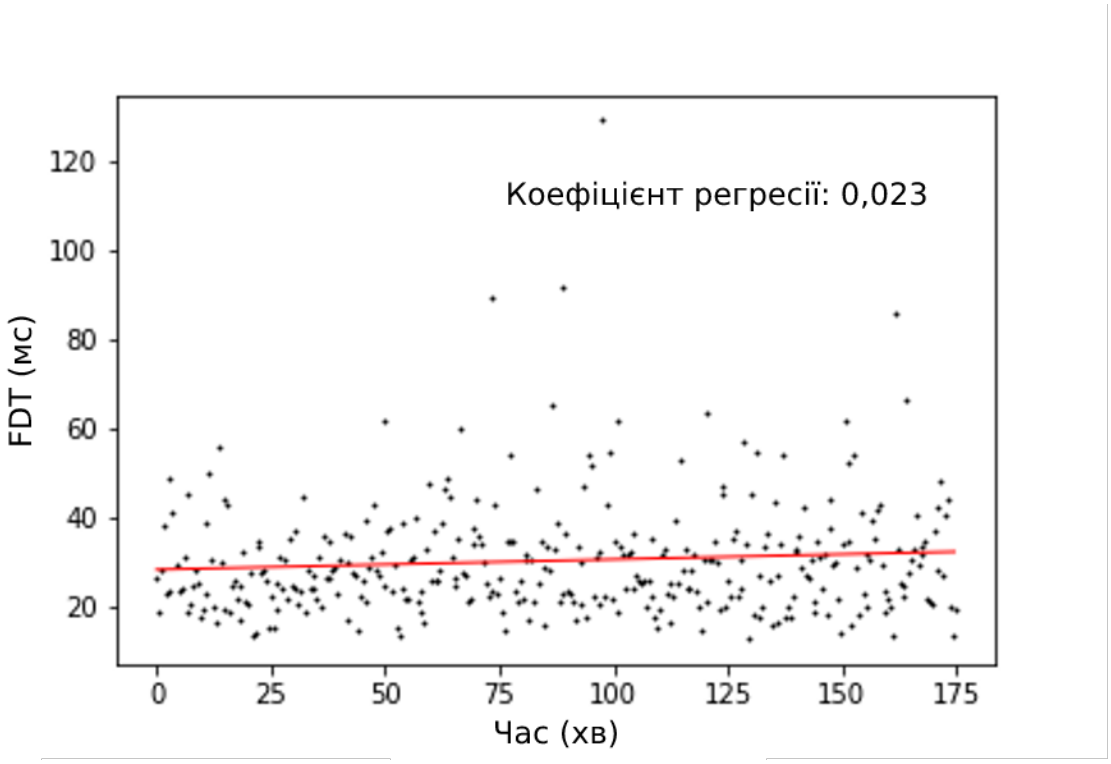


Рис. 3.3. Приклад лінійної регресії для метрики FDT

На рис. 3.4 показано застосування лінійної регресії для метрики JFC для першого експерименту без перезапуску застосунків. Ця метрика може бути застосована як для визначення наявності старіння, так і для більш детального вивчення старіння розглядаючи причини виникнення пропущених чи затриманих кадрів, що більш детально обговорюється в розділі 3.2.

В таблиці 3.2 вказані коефіцієнти лінійної регресії для кожної з виміряних метрик в кожному експерименті. Знак «+» вказує, що спостерігається старіння в експериментальних даних, а знак «-» вказує на те, що старіння не спостерігається. В 5 з 8 тестів тренди метрик ALT та FDT співпадають. В 4 з 8 тестів тренди метрик ALT та JFC співпадають. В 4 з 8 тестів тренди метрик FDT та JFC співпадають.

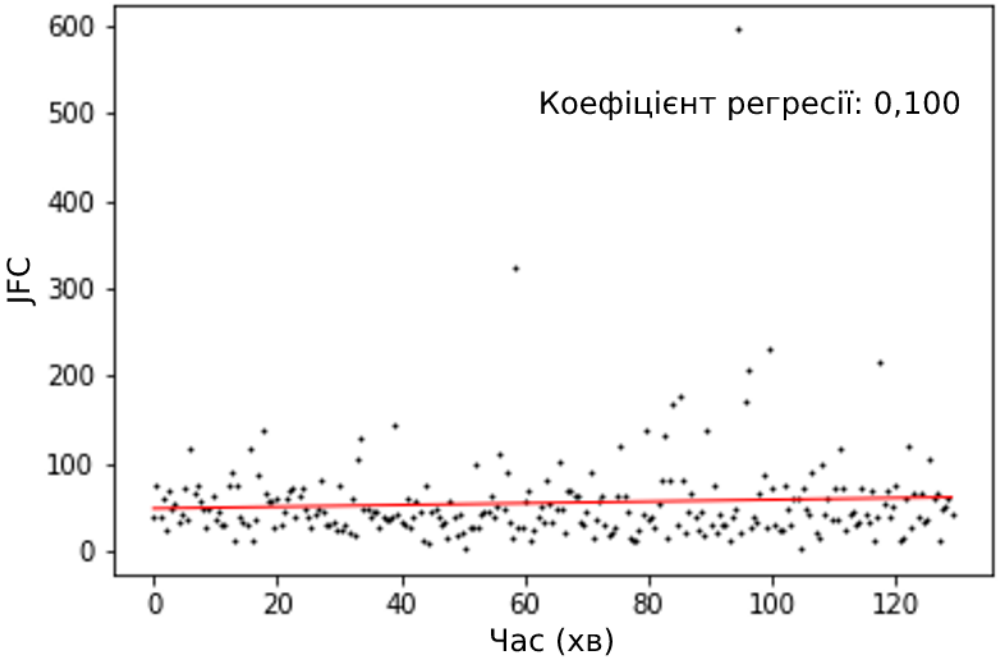


Рис. 3.4. Приклад лінійної регресії для метрики JFC

Таблиця 3.2.

Коефіцієнти лінійної регресії для виміряних метрик

Метрики	**Тестові випадки з примусовими перезапусками застосунків**				**Тестові випадки без примусових перезапусків застосунків**			
	1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
ALT	+0.03	+0.038	-0.275	-0.049	-0.022	+0.106	-0.664	+0.565
FDT	-0.02	+0.023	-0.026	+0.009	+0.009	+0.004	-0.01	+0.015
JFC	+0.1	-0.039	-0.005	+0,005	+0.1	+0.018	-0.065	-0.008
Avg PSS	+4.2	+6.2	+8.2	+5.5	+22.5	+17.8	+17.3	+22.5
Total PSS	+640	+532	+854	+577	+3411	+1939	+2889	+1493

В усіх випадках сумарний (Total) PSS та середній (Avg) PSS показують стабільний тренд зростання, що можна побачити на візуалізації виміряних значень PSS (Рис. 3.5) одного із тестових випадків. Крім того, спостерігається значна відмінність виміряних значень PSS для тестових випадків із перезавантаженнями та без перезавантажень застосунків, що підтверджує залежність використання оперативної пам’яті від сценарію використання мобільного пристрою. В даному

випадку мова йде про те, що часті перезавантаження мобільних застосунків, хоч і не реалістичні в контексті виконаного експерименту, однак мають вплив на зменшення використання оперативної пам’яті, що в свою чергу має збільшувати значення TTRE. Тобто, результати виконаних експериментів свідчать про те, що перезавантаження застосунків може зменшити швидкість процесу старіння та відтермінувати настання відмов через виснаження ресурсів.

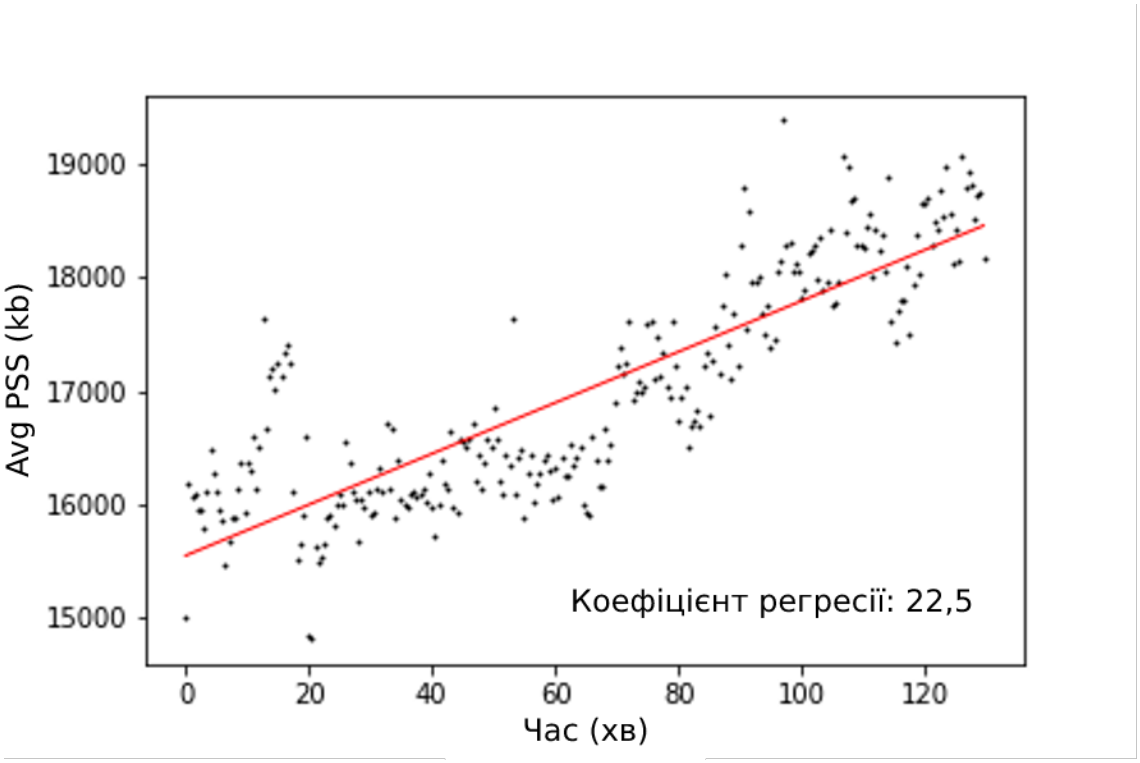


Рис. 3.5. Приклад застосування лінійної регресії для метрики PSS в експерименті без примусових перезавантажень користувацьких застосунків

Оскільки серед усіх розглянутих метрик не було знайдено сильних кореляцій, то зосереджено увагу на кореляціях між метриками кадрів та метриками сумарного та середнього PSS, детальну інформації для усіх тестових випадків подано в таблиці 3.3. Існує незначна негативна кореляція (-0.3±0.05) між JFC та Total PSS. Припускається, що це через те, що збільшення обсягу використовуваної пам’яті процесами зменшує кількість помилок кадрів через недостатність пам’яті. Потрібно продовжити дослідження окремих користувацьких застосунків та системних процесів для визначення кореляцій між метриками кадрів та використання пам’яті.

Таблиця 3.3.

Коефіцієнти кореляції Спірмена між виміряними метриками та трендами PSS

Метрики	**Тестові випадки з примусовими перезапусками застосунків**				**Тестові випадки без примусових перезапусків застосунків**			
	1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
ALT та Total / Avg PSS	-0.03; 0.06;	-0.001; 0.04;	-0.06; -0.06;	-0.03; -0.01;	0.1; -0.05;	0.09; 0.07;	-0.05; -0.11;	-0.05; -0.07;
FDT та Total / Avg PSS	-0.03; -0.11;	0.18; 0.04;	0.01; -0.16	-0.04; -0.09;	0.09; 0.04;	0.03; 0.08;	-0.01; -0.08;	0.11; -0.05;
JFC та Total / Avg PSS	-0.23; 0.01;	-0.3; -0.03;	-0.33; 0.03;	-0.2; 0.05;	-0.13; -0.04;	-0.26; 0.09;	-0.14; -0.11;	-0.27; 0.05

Оцінка метрик з допомогою методу Манна-Кендала у EXP2 (конфігурація експерименту описана в додатку Б) показує, що у 4 із 12 тестових випадків спостерігається тренд збільшення значень FDT, що показано в таблиці 3.4. В свою чергу, тільки в 1 із 12 тестових випадків спостерігався тренд збільшення метрики JFR. Метрика ALT взагалі не показала статистично значимого тренду старіння.

Таблиця 3.4.

Результати тесту Манна-Кендала та процедури Сена для тестових випадків EXP2

Тестовий випадок	**Фактори**		**FDT**	
	Застосунки	**Використання**	**P-значення**	**Нахил Сена**
1	Flutter	Безперервне	0.302399	-0.003416
2	**Flutter**	**Безперервне**	**0.002441**	**0.017006**
3	Flutter	Безперервне	0.224849	-0.004758
4	Flutter	Із паузами	0.763258	-0.003173
5	**Flutter**	**Із паузами**	**0.013110**	**0.019033**
6	Flutter	Із паузами	0.744591	0.002650
7	Native	Безперервне	0.722181	-0.000203

Тестовий випадок	Фактори		FDT	
	Застосунки	**Використання**	**P-значення**	**Нахил Сена**
8	Native	Безперервне	0.169906	-0.000674
9	Native	Безперервне	0.072456	-0.001184
10	Native	Із паузами	0.000492	-0.002452
11	**Native**	**Із паузами**	**0.000118**	**0.003472**
12	**Native**	**Із паузами**	**0.032811**	**0.002924**

Метрика FDT може бути використана в моделях омолодження, які враховують активність користувача, оскільки ця метрика дозволяє охарактеризувати патерни поведінки користувача, що можна чітко побачити на рис. 3.6. Крім того, оцінка метрики FDT показує, що можливі випадки, коли на окремих інтервалах активності користувача не спостерігається жодного тренду. Тобто, у реальних умовах використання, можна визначати як загальний тренд, так і індивідуальний для окремих періодів використання, щоб мати змогу аналізувати вплив активності користувача на процес старіння.

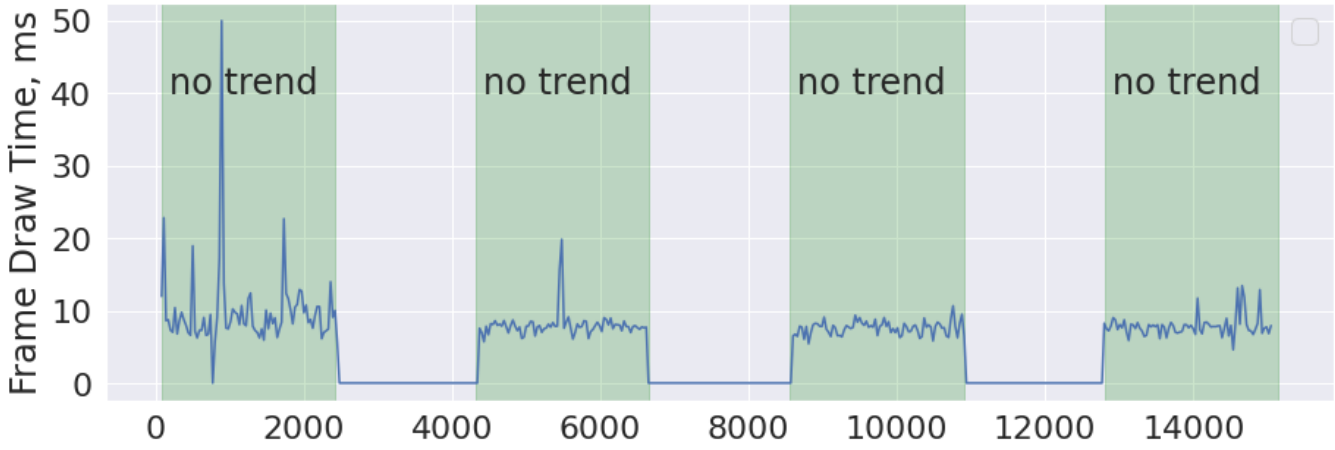


Рис. 3.6. Тренди метрики FDT у випадку сценарію використання із паузами генерування робочого навантаження

Отже, результати проведених експериментів підтверджують ефективність та доцільність використання метрики FDT для виявлення старіння в ОС Android, а метрики JFC та ALT можуть використовуватись додатково для більш точної оцінки

критичності рівня старіння в системі. Метрика FDT може бути застосована для оцінки старіння в різних сценаріях використання мобільних застосунків, а також, вона може бути використана для обчислення інтенсивностей переходів між станом активного використання мобільного пристрою користувачем та станом очікування пристрою, які необхідні для моделей старіння та омолодження описаних у попередньому розділі. В свою чергу, виокремлення різних порогових значень метрики FDT дозволяє визначати різні стани продуктивності GUI та використовувати цю метрику в запропонованих моделях на основі ланцюгів Маркова, які враховують різні рівні старіння.

3.2. Дослідження процесів старіння програмного забезпечення в операційній системі Android з урахуванням впливу сценаріїв використання та типів застосунків

Розроблення ефективних методів омолодження ПЗ вимагає розуміння чинників, що мають вплив на процес старіння та вразливих до старіння елементів системи Android. Тому, важливими науково-прикладними завданнями є виявлення факторів старіння та аналіз процесу старіння в ОС Android з точки зору системного та користувацького ПЗ.

Більшість емпіричних досліджень старіння в ОС Android розглядає цей процес на рівні ОС без урахування старіння окремих користувацьких застосунків. Такі дослідження обмежують різноманітність сценаріїв використання мобільних застосунків, зокрема, шляхом їх регулярного перезавантаження, щоб уникнути накопичення помилок, що призводять до витоків пам’яті в окремих користувацьких застосунках. Результати таких досліджень [89, 91] дозволяють ідентифікувати найбільш вразливі і ресурсномісткі процеси системи. Такими процесами в ОС Android є:

- System Server – це Java процес який запускається під час завантаження Android та ініціалізує Android Framework. Він містить більшість системних служб, таких як Activity Manager, яка управляє життєвим циклом застосунків та їх Activity, та Package Manager, яка управляє установленими пакетами та дозволами безпеки. Цей процес також регулює доступ до системних ресурсів;

- System UI – це процес, який компонує області екрана для відображення сповіщень, повідомлення про стан пристрою та кнопок навігації за допомогою системних панелей;

- Surface Flinger – це процес, що отримує шари вікон (поверхні) з різних джерел (включно із System UI), об'єднує їх та відображає на дисплеї мобільного пристрою.

На процес накопичення помилок старіння ПЗ, що призводить до виникнення ефектів старіння, які можуть бути виявлені з допомогою метрик старіння, мають вплив різні чинники. Виявлення вагомих факторів старіння є важливим для пошуку рішень мінімізації їх впливу на погіршення показників продуктивності та надійності ПЗ. Дослідження старіння ПЗ в ОС Android різних авторів [89, 91] враховували фактори, які перелічено в табл. 3.5.

Таблиця 3.5

Фактори старіння ПЗ в ОС Android

Фактор	**Приклади**
Модель та технічні характеристики мобільного пристрою	Huawei P8 HTC One M9 LG Nexus Samsung S6 Edge
Версія ОС Android	5.0, 6.0, 7.0
Розробник мобільних застосунків	Застосунки Android та Google Застосунки Huawei Застосунки Google Play від сторонніх розробників
Частота перезапусків користувацьких застосунків	Перезапуск кожні 5 секунд Перезапуск кожні 60 секунд Без перезапуску
Події введення	Перемикання між застосунками Події сенсорного введення, прокручування, навігація

Фактор	**Приклади**
	Без подій введення
Обсяг файлового сховища	Заповнення файлового сховища на 90% Звичайний обсяг вільного простору

В дослідженні [89] розглядаються мобільні пристрої різних виробників та версії ОС Android, зокрема, 5.0, 6.0, 7.0. Результати аналізу впливу цих факторів свідчать про те, що немає значної відмінності у виникненні старіння на мобільних пристроях від різних виробників. В свою чергу, аналіз різних версій Android показав, що із кожною новою версією не спостерігається покращення значень метрик старіння. Дослідження та аналіз факторів старіння в іншій роботі [91] показали, що застосунки сторонніх розробників вразливіші до старіння ПЗ, ніж стандартні системні застосунки Android та Google, а також, спостерігається залежність впливу технічних характеристик різних моделей пристроїв та активності генерування робочого навантаження на процес старіння ПЗ.

Оскільки в попередніх дослідженнях старіння значна увага приділяється його проявам на рівні ОС, тому важливо розглянути старіння різного рівня, зокрема рівня користувацьких застосунків. Крім того, в даному розділі пропонується розглянути такі нові рівні факторів, які б дозволили поглибити розуміння впливу користувацьких застосунків та активності використання мобільного пристрою на процес старіння ПЗ.

Важливим ресурсом системи, який використовується системним та користувацьким ПЗ є оперативна пам’ять. На відміну від частки завантаженості CPU, зростання використання RAM частіше може вказувати на наявність процесів старіння, оскільки навантаження на центральний процесор не обов’язково означатиме відмову мобільного пристрою внаслідок старіння, а малий об’єм вільної оперативної пам’яті може призводити до відмов типу «Out of Memory». В ОС Android реалізовано два механізми управління низьким об’ємом вільної оперативної пам’яті [108]: KSD та LMKD. На рівні процесів ОС використовується LMKD, який примусово завершує їх

роботу і звільняє використовувані ресурси. Для прийняття рішення про те, який процес може бути завершений, LMKD використовує так звану оцінку «oom_adj_score» для пріоритезації процесів системи в контексті відмов «out of memory». Кожному процесу, що описує системні сервіси чи прикладні застосунки, LMKD присвоює певне цілочисельне значення «oom_adj_score». Процеси з найвищою оцінкою примусово завершуються першими, а процеси із найнижчою оцінкою вважаються критично важливими для життєдіяльності системи. Відповідно до можливих значень «oom_adj_score» можна виділити ряд категорій процесів ОС Android із різними характеристиками (табл. 3.6).

Таблиця 3.6

Категорії процесів ОС Android у відповідності до значень оцінки «oom_adj_score»

Назва категорії	**Опис**
Фонові застосунки (cached, serviceb)	Не активні в даний момент застосунки. Завершуються першими в порядку зменшення присвоєного їм рейтингу.
Попередній застосунок (prev)	Нещодавно запущений застосунок, який з більшою імовірністю користувач знову відкриватиме
Домашній застосунок (home)	Програма запуску, закриття якої призведе до зникнення шпалер на головному екрані
Сервіси (servicea)	Сервіси застосунків, які запускаються і контролюються в першу чергу застосунками і можуть виконувати, наприклад, завантаження даних на сервер чи інші складні обчислення.
Помітні застосунки (percept)	Помітні для користувача застосунки, але не знаходяться на передньому плані, наприклад, системні кнопки навігації.
Застосунок переднього плану (fore, vis)	Застосунок, що відображається і використовується безпосередньо користувачем в даний момент. Його примусове завершення виглядатиме для користувача як помилка виконання і є індикатором того, що пристрій працює погано.

Назва категорії	**Опис**
Стійкі сервіси (pers, persvc)	Основні сервіси мобільного пристрою, які управляють телефонним, wifi зв’язком і т.д..
Системні процеси (system)	У випадку завершення цих процесів телефон може перезавантажитись.
Нативні процеси (native)	Низькорівневі процеси системи, зокрема KSD.

Результати виконання експериментальних досліджень EXP1 та EXP2 (див. додаток Б) дозволили виконати аналіз старіння системних процесів та користувацьких застосунків.

Виконання експериментального етапу дослідження дозволило отримати дані про використання процесами пам’яті системи, а також про динаміку зміни продуктивності різних процесів та застосунків. Розподіл використання пам’яті процесами різних категорій (Рис. 3.7) показує, що більша частина оперативної пам’яті використовується користувацькими застосунками та сервісами: pers, fore, vis, home, prev, cached. В цьому випадку важливо ідентифікувати групи процесів, в яких спостерігається стабільний тренд збільшення використання оперативної пам’яті, що може свідчити як про наявні в них помилки пов’язані із витоками пам’яті, так і про природнє збільшення використовуваної пам’яті для користувацьких потреб.

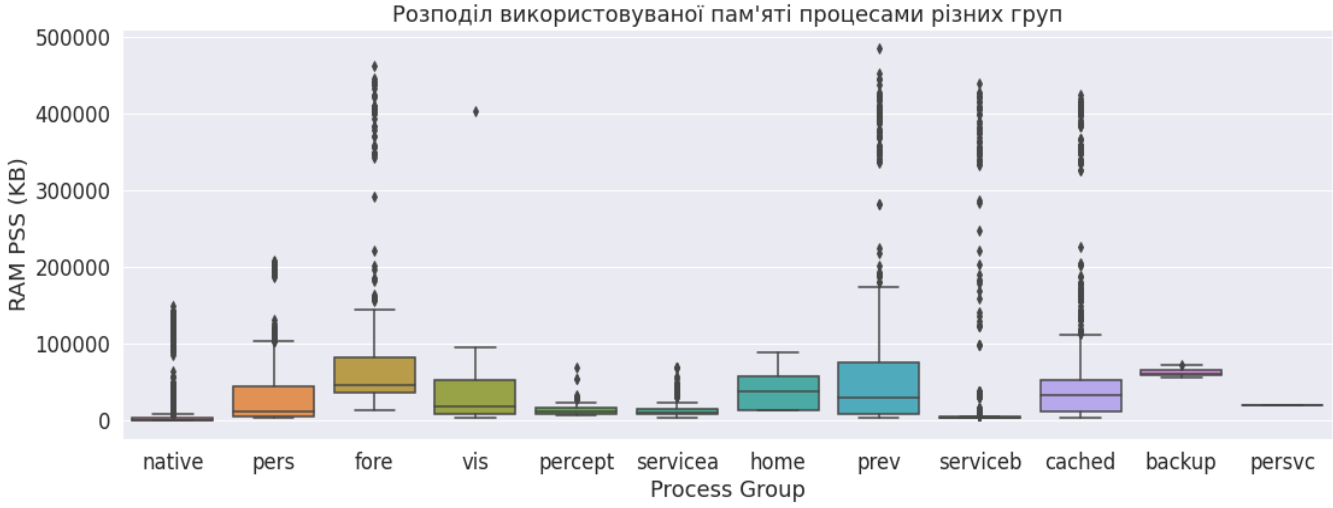


Рис. 3.7. Розподіл використання пам’яті процесами різних категорій

На рис. 3.8 показано кількість трендів збільшення використання пам’яті процесами різних груп. Визначення тренду виконано з допомогою тесту Манна-Кендала та процедури Сена для визначення нахилу тренду. Статистична значимість для виявлення тренду визначена як 90% (α=0.1). Тобто, якщо $P < 0.1$, а значення нахилу більше нуля, тоді спостерігається тренд збільшення. Результати виявлення трендів показують, що майже у всіх випадках тестів спостерігається стабільне збільшення використання пам’яті користувацькими застосунками (vis) та збільшення об’єму кешованих застосунків (cached, backup), що свідчить про збільшення навантаження на систему і необхідність в майбутньому використання LMK для звільнення пам’яті для нових процесів. Недоліком такої ситуації є те, що прийняття рішення про завершення роботи користувацьких процесів відбувається на стороні системи, що в деяких випадках може бути критично для користувача, який очікує завершення виконання своїх задач. Отримані в цьому експерименті результати підтверджують доцільність застосування перезавантаження користувацьких застосунків, що спостерігалось під час аналізу метрики PSS в попередньому експерименті. Крім того, категорія pers, в яку входять процеси системних сервісів, також зазнає старіння в 7 з 8 тестів.

Рис. 3.8. Тренди використання оперативної пам’яті процесами різних груп

Проаналізувавши отримані результати і беручи до уваги роботу LMKD, можна виділити дві умовні групи процесів з точки зору старіння ПЗ:

1. Системні процеси (native, system, pers, persvc, home, percept) – можна застосувати процедуру омолодження до основних сервісів системи;

2. Користувацькі застосунки (fore, vis, prev, servicea, cached, serviceb) – ефективність протидії старінню в цих процесах полягає в першу чергу в мінімізації кількості помилок, що призводять до старіння, шляхом застосування ефективних структур даних, алгоритмів, архітектурних шаблонів та механізмів управління ресурсами.

Аналіз «oom_adj_score» також дозволяє зробити висновок про доцільність використання цієї метрики для визначення стану системи під час планування чи виконання процедури омолодження ПЗ. Визначення кількості користувацьких застосунків, що відображаються на передньому плані та сервісів, що виконуються в фоновому режимі, дозволяє приймати зважене рішення про доцільність виконання процедури омолодження, щоб уникнути перешкоджання використання пристрою користувачем через перезавантаження тих чи інших процесів.

Для аналізу старіння системних процесів виконано підрахунок кількості трендів збільшення використання пам’яті процесами та збільшення тривалості роботи збирача сміття серед всіх виконаних тестів з допомогою методу Манна-Кендала. Визначення трендів метрик GC Pause Time та GC Duration дозволить дати більш точну оцінку явищу старіння в системних процесах. GC Pause Time – це затримка виконання процедури збирача сміття, а GC Duration – це загальна тривалість виконання процедури збирача сміття. Обидві метрики генеруються системним процесом ART [109] при умовах, що час перевищує 5 мс. або 100 мс. відповідно.

На рис. 3.9 зображено тільки ті процеси, в яких спостерігається старіння більше ніж в половині виконаних тестів і середній обсяг використовуваної оперативної пам’яті перевищує 20 Mb. Процес system показує найбільшу вразливість до старіння. В інших процесах також регулярно спостерігаються тренди збільшення використання пам’яті, однак відсутні тренди тривалості чи затримки роботи збирача сміття.

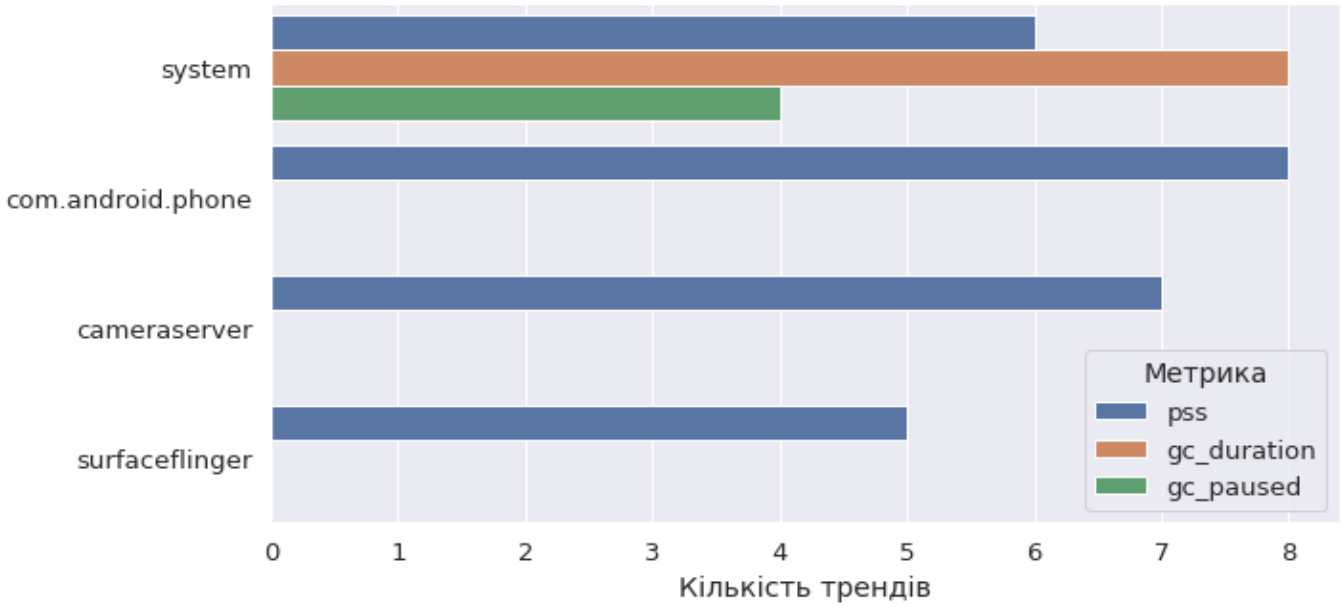


Рис. 3.9. Результати EXP1. Системні процеси Android, які найчастіше проявляють статистично значимі тренди збільшення використання пам’яті та збільшення тривалості роботи GC

Серед розглянутих системних процесів, в яких спостерігаються тренди старіння, не виявлено сильних кореляцій із метриками відображення кадрів. Спостерігається незначна негативна кореляція між PSS cameraserver та JFR, яка коливається між -0,12 і -0,19 для 7 із 8 стресових тестів. Така кореляція може пояснюватися тим, що збільшення об’єму пам’яті, яку використовує процес, дозволяє зменшити кількість затримок відображення кадрів. Припускається, що наявність чи відсутність кореляцій може залежати від конфігурації стресового тесту. Зокрема, в цій роботі стресовому навантаженню піддавався застосунок камери, який використовувався приблизно 0,2 від всього часу виконання тестів. Внаслідок цього спостерігається постійний тренд збільшення використання пам’яті системним сервісом cameraserver та кореляція із метрикою частки пропущених кадрів.

Аналіз результатів експерименту EXP2 (див. додаток Б) показав, що тренди старіння спостерігаються в процесах (рис. 3.10) system, com.android.systemui та surfaceflinger. Крім того, у двох тестових випадках, де використовувались Flutter застосунки, спостерігається старіння процесу mm-qcamera-daemon, а три тестові випадки з використанням native застосунків показав старіння в процесі cameraserver.

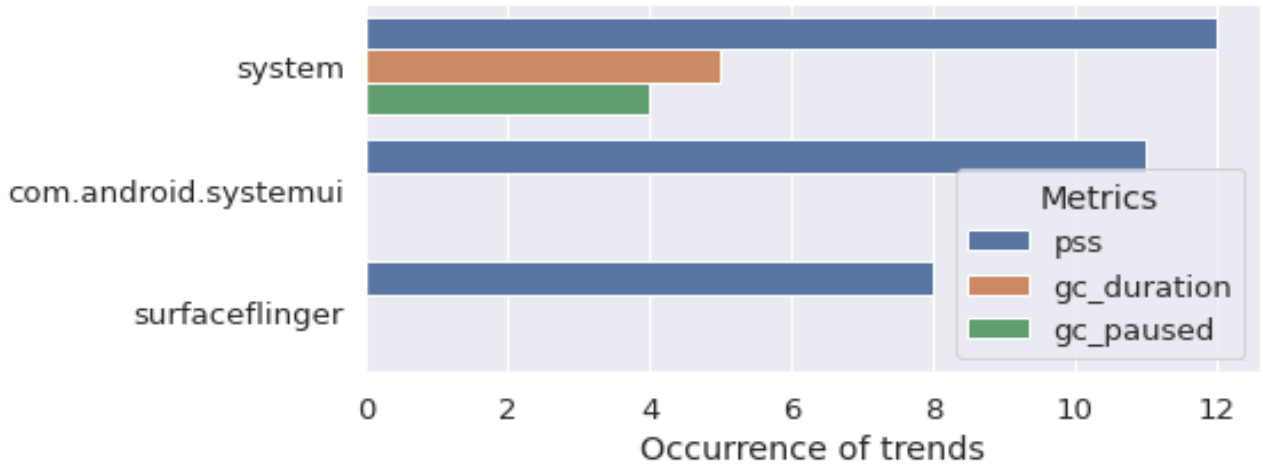


Рис. 3.10. Результати EXP2. Системні процеси Android, які найчастіше проявляють статистично значимі тренди збільшення використання пам’яті та збільшення тривалості роботи GC

Таким чином, аналіз системних процесів дозволяє зробити висновок, що в ОС Android ефектам старіння піддаються процеси system та surfaceflinger, а також процеси, які в більшій мірі залежать від діяльності користувача та роботи користувацьких застосунків. Наприклад, com.android.phone, cameraserver, audioserver та інші. Тому, для реалізації засобів протидії старінню чи омолодження ПЗ важливо дослідити і враховувати вплив різних процесів в залежності від конкретних умов використання мобільного пристрою.

Запропоновані в другому розділі метрики відображення кадрів графічного інтерфейсу користувача для виявлення старіння в системі можуть бути застосовані також і для виявлення старіння в межах окремих застосунків. Серед більшості користувацьких застосунків та процесів спостерігаються тренди збільшення використання пам’яті, що є природньо. Трендів збільшення тривалості роботи збирача сміття не виявлено. В одному випадку з восьми було виявлено тренди збільшення метрик FDT та JFR для процесу com.android.chrome. Отримані результати можуть пояснюватись короткою тривалістю виконання стресових тестів.

Для уникнення старіння в користувацьких застосунках важливо розуміти причини затримки кадрів та збільшення часу їх відображення. Команда командного рядка `adb shell dumpsys gfxinfo` дозволяє отримувати описані метрики, а

також показники, що вказують на можливі причини затримок кадрів чи тривалого відображення кадрів [110]:

- Missed Vsync – пропущені вертикальні синхронізації відображення кадру і оновлення дисплею пристрою;
- High input latency – висока затримка введення для відображення наступного кадру;
- Slow UI thread – повільна робота потоків UI;
- Slow bitmap uploads – повільне завантаження растрових зображень;
- Slow issue draw commands – повільне виконання команд малювання.

Виконані експерименти (рис. 3.11) показують, що найбільше кадрів втрачається внаслідок перевантаження основних потоків відображення UI, пропусків вертикальної синхронізації, а також внаслідок повільного виконання команд малювання. Таким чином, можна зробити висновок, що зменшення впливу розглянутих чинників дозволить зменшити ефекти старіння загалом, тому важливо враховувати наступні рекомендації: уникати виконання складних обчислень в основному потоці UI, використовувати коректні методи малювання графіки, растрових зображень та інше.

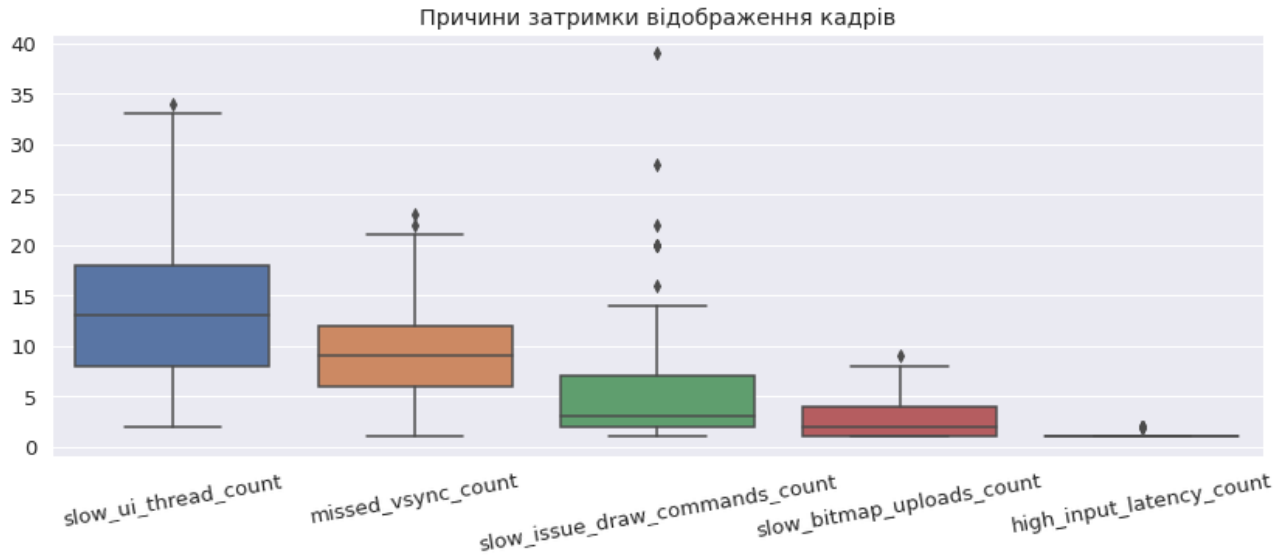


Рис. 3.11. Причини затримки відображення кадрів UI для виконаних експериментів

В дизайні експериментального дослідження EXP2 (див. додаток Б) враховано такі фактори [12] як сценарії використання мобільного пристрою та набори користувацьких застосунків. Повний опис конфігурації тестових випадків подано в додатку Б.

Оскільки попередні дослідження застосунків від сторонніх розробників підтверджують їх вразливість до ефектів старіння ПЗ, то важливо дослідити застосунки, які використовують крос-платформові фреймворки, такі як Flutter [111]. Крос-платформові фреймворки є популярні серед розробників через відносну простоту їх використання та швидкість розроблення для відразу двох мобільних ОС Android та iOS. Більшість вихідного коду таких проектів є спільним для обох систем. Однак, незалежність спільного коду від native коду Java/Kotlin Android SDK, особливості механізмів взаємодії спільного коду із сервісами ОС, а також, архітектура крос-платформових фреймворків, може мати вплив на процес старіння ПЗ. Наприклад, крос-платформовий фреймворк Flutter використовує Android Activity [112] як контейнер для відображення всього UI та всіх екранів застосунку, на відміну від native застосунків, які можуть оперувати багатьма екземплярами Activity.

Під час виконання експерименту EXP2 враховано генерування робочого навантаження для двох наборів різних типів користувацьких застосунків. Набір стандартних native застосунків Android та Google: com.google.android.youtube, com.android.chrome, com.android.camera, com.google.android.apps.photos, com.google.android.apps.maps. В противагу цим застосункам обрано інший набір від сторонніх розробників, реалізованих з допомогою крос-платформової технології Flutter: com.reflectlyApp, com.hamilton.app, com.ebay.motorsapp, com.spotlightsix.zentimerlite2, com.mgmresorts.mgmresorts. Обрані Flutter-застосунки мають кількість завантажень в Google Play маркеті понад 100000 та 500000.

Для аналізу факторів застосовано метод візуалізації з допомогою діаграми розмаху. Діаграма розмаху для метрики FDT (рис. 3.12) показує, що Flutter застосунки серед усіх виконаних тестових випадків показує значно більшу тривалість відображення кадрів, ніж стандартні додатки Google та Android. В той час, коли native

застосунок відображає кадр менше, ніж за 25 мс., Flutter застосунок може відображати кадр UI більше, ніж за 50 мс.

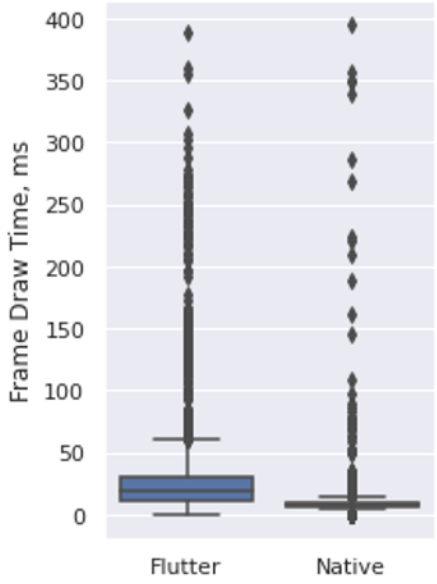


Рис. 3.12. Діаграма розмаху значень метрики FDT для різних типів користувацьких застосунків

Діаграма розмаху значень метрики JFR (рис. 3.13) підтверджує, що частка пропущених чи затриманих кадрів у тестових випадках із Flutter застосунками може перевищувати 50%, а в більшості випадків близько 25%. В свою чергу, стандартні застосунки показують частку пропущених кадрів не більше 30%. Розподіл значень метрик JFR та FDT свідчать про вплив тривалості відображення кадрів на збільшення частки пропущених кадрів, а також, меншу стійкість до старіння Flutter застосунків.

Розгляньмо ще один фактор, який може мати вплив на процес старіння ПЗ. Особливістю мобільних пристроїв, що відрізняє їх від більшості програмно-апаратних систем, таких як ПК чи веб-сервери, є їх інтервальне використання з різною інтенсивністю. Запропоновані в другому розділі моделі старіння та омолодження враховують активність використання мобільного пристрою користувачами, а досліджені метрики відображення кадрів дозволяють отримати необхідну статистичну інформацію. Разом з цим, важливо дослідити процес старіння в ОС

Android з урахуванням сценаріїв використання з непостійним генеруванням робочого навантаження.

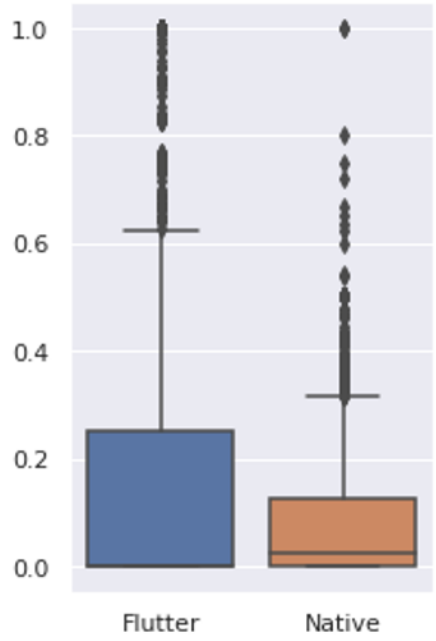


Рис. 3.13. Діаграма розмаху значень метрики JFR для різних типів користувацьких застосунків

У виконаному досліджені EXP2 розглянуто два сценарії використання мобільних застосунків. Перший сценарій полягає у неперервному використанні набору застосунків між якими відбувається постійне перемикання та інші події введення і навігації. Другий сценарій імітує використання мобільного пристрою із паузами між інтервалами неперервної генерації робочого навантаження. Наприклад, під час виконання одного тестового випадку протягом чотирьох годин робоче навантаження генерується неперервно 40 хвилин із 30-хвилинними паузами. Такий дизайн експерименту дозволить оцінити динаміку старіння системи в кожному окремому випадку, а також порівняти поведінку системи в різних інтервалах активності користувача. Зокрема, важливо оцінити зміни продуктивності користувацького інтерфейсу з допомогою метрик відображення кадрів, а також, охарактеризувати старіння з допомогою метрики PSS для процесу system_server, який

згідно попереднього аналізу, є вразливим до старіння та вагомим показником старіння.

Діаграма розмаху для метрики FDT (рис. 3.14) показує, що сценарії використання мобільного пристрою, які були реалізовані в даній роботі, мають незначний вплив на тривалість відображення кадрів. Спостерігається підняття значення першого квартиля для сценарію використання із неперервним генеруванням робочого навантаження в порівнянні із сценарієм з паузами активності користувача.

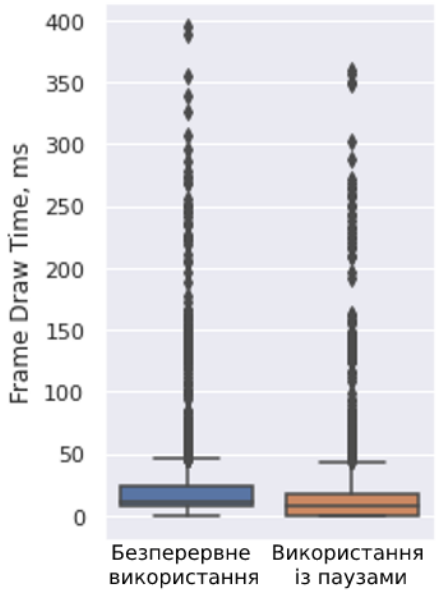


Рис. 3.14. Діаграма розмаху значень метрики FDT для різних сценаріїв використання мобільного пристрою

Діаграма розмаху для метрики JFR (рис. 3.15) свідчить про вплив фактору активності користувача на збільшення частки пропущених кадрів. Перший квартиль в обох випадках знаходиться на рівні 0%, а третій квартиль у випадку сценарію із безперервним робочим навантаженням показує 20%, що майже втричі більше, ніж у сценаріях із паузами. Це свідчить про існування явища старіння в тому сенсі, що користувач, який активніше використовує мобільний пристрій за той самий час, може спричиняти пришвидшення процесу старіння.

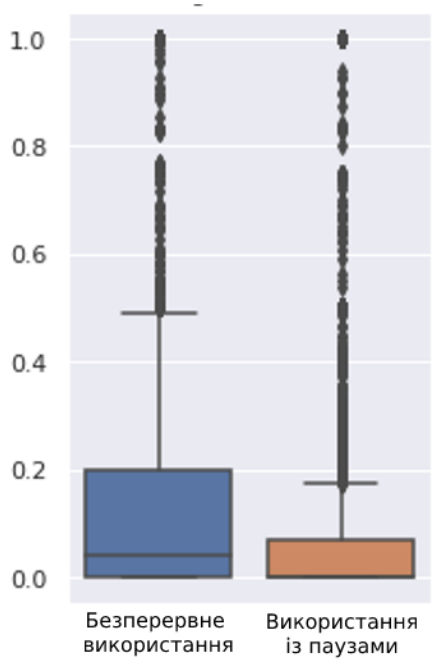


Рис. 3.15. Діаграма розмаху значень метрики JFR для різних сценаріїв використання мобільного пристрою

Крім оцінки продуктивності роботи GUI для сценаріїв використання важливо проаналізувати явище старіння з точки зору використання пам’яті таким системним процесом як system_server. В той час як FDT може не показувати трендів збільшення часу відгуку інтерфейсу, в system_server PSS може спостерігатись стабільний тренд збільшення («increasing» інтервали на рис. 3.16) використання пам’яті протягом генерування робочого навантаження. Це свідчить про те, що стабільне збільшення використання пам’яті, навіть без погіршення продуктивності GUI, може збільшувати час до відмови через виснаження ресурсів, тобто час до відмови старіння. Крім того, тренд збільшення використання пам’яті може спостерігатись навіть в ті моменти, коли користувач не використовує мобільний пристрій. Це може пояснюватись тим, що в пам’яті все ще залишаються процеси, які виконуватимуться у фоновому режимі. Таким чином, отримані результати підтверджують важливість виконання процедури омолодження, а саме, очищення процесів та скидання даних для уникнення переповнення пам’яті.

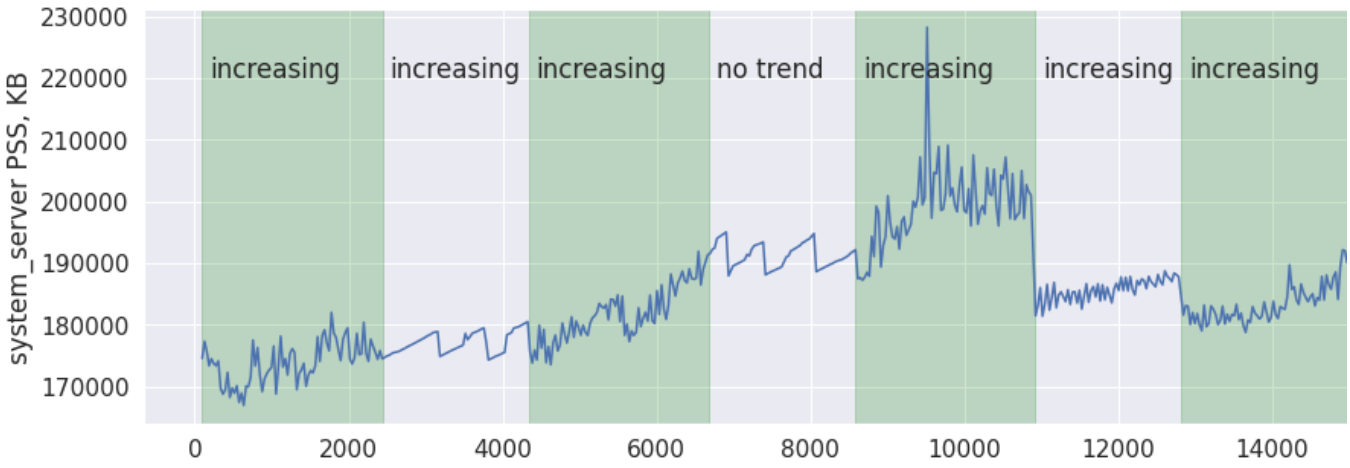


Рис. 3.16. Тренди використання оперативної пам’яті процесом system_server під час використання мобільного пристрою із паузами

Для порівняння впливу факторів на метрики FDT та JFR виконано однофакторний дисперсний аналіз (табл. 3.7). Обидва фактори мають вагомий вплив на вимірювані метрики відображення кадрів GUI. Тип користувацьких застосунків має сильніший вплив, ніж сценарій використання.

Таблиця 3.7.

Результати застосування однофакторного дисперсного аналізу для експериментального дослідження EXP2

Метрика	**Фактор**	**df**	**SS**	**MS**	**F**	**P-значення**
FDT	Застосунки	1.0	5.0e+5	504039	432	1.5e-92
	Залишок	5849	6.8e+6	1168		
	Сценарії	1.0	7.2e+4	71580	58	3.6e-14
	Залишок	5849	7.3e+6	1242		
JFR	Застосунки	1.0	20.8	20.8	328	2.5e-71
	Залишок	5849	371.8	0.06		
	Сценарії	1.0	7.9	7.9	120	1.0e-27
	Залишок	5849	384.8	0.07		

3.3. Модель факторів старіння програмного забезпечення для операційної системи Android

Для застосування комплексної аналітичної моделі старіння та омолодження [6] для прогнозування часу омолодження важливим завданням є обчислення інтенсивностей переходів між станами, а також, визначення стану системи в конкретний момент часу. Для вирішення цієї задачі пропонується наступна модель факторів старіння, яку можна представити множинами метрик старіння та метрик для відповідних факторів старіння. Модель факторів старіння враховує як відомі метрики старіння ALT, PSS та GCT, так і нові метрики UI [3], що запропоновані в розділі 3.1.

Можна виділити наступні множини метрик:

- $M_{Aging} = \{ALT, FDT, JFR, PSS, GCT\}$ – метрики старіння;
- $M_{Usage} = \begin{Bmatrix} ALT, FDT, \\ частка\ процесів\ користувача, \\ частка\ використання\ CPU, \\ частка\ суми\ PSS \end{Bmatrix}$ – метрики активності використання мобільного пристрою;
- $M_{Battery} = \begin{Bmatrix} поточний\ рівень\ заряду, \\ прогнозований\ час\ розряду \end{Bmatrix}$ – метрики стану заряду батареї.

Сформовані множини часових рядів вимірюваних метрик необхідні для визначення інтенсивностей переходів між можливими станами системи згідно моделі прогнозування часу омолодження стану, а саме, для обчислення значень матриці Q в формулі (2.2). В таблицях 3.8-3.10 наведено умови та наближені значення вимірюваних метрик, при яких можливе перебування в тому чи іншому стані. Необхідне детальне дослідження запропонованих порогових значень та ефективність їх застосування в потенційних методах омолодження.

Для забезпечення плавної роботи GUI необхідно відображати 60 (≤ 16 мс.) кадрів за секунду, що є умовою перебування системи в стані «Young», а при кількості кадрів менше 15-ти (> 66 мс.) фіксується відмова старіння і стан «Failure». Порогові значення метрики PSS визначені відносно загального обсягу доступної оперативної пам’яті. Порогові значення метрики ALT визначені на основі офіційної документації

[113], де «гарячий» (до 1,5 с.) і «теплий» (до 2 с.) запуски можуть характеризувати стан «Young», «холодний» (до 5 с.) – стан «Aging», а тривалість запуску Android Activity більше 5 с. та 10 с. характеризує стан «Old» та «Failure» відповідно. У випадку GCT нижнє значення 100 мс. визначає подію, яка є індикатором виникнення затримок, тому пропонуються більші порогові значення для різних станів старіння.

Таблиця 3.8.

Метрики та їх пороги для визначення рівня старіння системи

Метрики		**Стан старіння системи**			
Назва	**Пріоритет**	**Young**	**Aging**	**Old**	**Failure**
FDT	Високий	≤ 16 мс.	> 16 мс. & ≤ 40 мс.	> 40 мс. & ≤ 66 мс.	> 66 мс.
PSS	Високий	≤ 20%	> 20 % & ≤ 80 %	> 80 % & ≤ 95 %	> 95 %
JFR	Середній	≤ 20%	> 20 % & ≤ 80 %	> 80 % & ≤ 95 %	> 95 %
ALT	Середній	< 2 с.	≥ 2 с. & < 5 с.	≥ 5 s & < 10 s	≥ 10 с.
GCT	Середній	≤ 100 мс.	> 100 мс. & ≤ 200 мс.	> 200 мс. & ≤ 300 мс.	> 300 мс.

Для обчислення інтенсивностей переходів між станами старіння (табл. 3.8) доцільно застосовувати методи для виявлення наявного позитивного тренду для кожного окремого часового ряду, який описує поведінку мобільного пристрою від моменту його увімкнення до вимкнення. Таким методом може бути тест Манна-Кандела. Якщо тренд існує, тоді, застосувавши регресійний метод, можна отримати криву, перетин якої із пороговими значеннями буде давати часові відрізки перебування системи в кожному із станів. Обчисливши середні значення для всіх отриманих часових відрізків та застосувавши формулу (2.3), можна отримати інтенсивності переходів. В свою чергу, виставлені пріоритети для метрик необхідні, щоб відкинути ті випадки, коли старіння спостерігається тільки в метриках з низьким пріоритетом. Наприклад, якщо тренд старіння спостерігається тільки в одній із метрик з середнім рівнем, тоді цей випадок можна відкинути і не враховувати в обчисленнях інтенсивностей.

У випадку визначення рівня активності використання мобільних пристроїв (табл. 3.9) метрики FDT та ALT можуть бути індикаторами стану «Active» при значеннях більших за 0, а частки процесів користувача, використання CPU та PSS при відносному значенні більшому за 20%.

Таблиця 3.9.

Метрики та їх пороги для визначення рівня активності використання мобільних пристроїв

Метрики		**Активність використання мобільного пристрою**	
Назва	**Вага**	**Sleep**	**Active**
FDT	1,0	0	> 0
ALT	0,5	0	> 0
Частка процесів користувача	0,5	< 20%	≥ 20%
Частка використання CPU	0,25	< 20%	≥ 20%
Частка PSS застосунків	0,25	< 20%	≥ 20%

Порогові значення метрик стану заряду батареї потребують більш детального дослідження і можуть бути як абсолютними значеннями (прогнозований час до повного розрядження), так і відносними (відсоток заряду батареї).

Таблиця 3.10.

Метрики та їх пороги для визначення стану заряду батареї

Метрики		**Стан заряду батареї**		
Назва	**Вага**	**High Power**	**Low Power**	**Off Power**
Прогнозований час розряду	1,0	> 1 години	≤ 1 години	0
Поточний рівень заряду	0,25	> 20 %	≤ 20 %	0%

У випадку метрик та порогів активності використання (табл. 3.9) та стану заряду батареї (табл. 3.10) для обчислення інтенсивностей потрібно враховувати частоту змін їх станів протягом всього циклу функціонування мобільного пристрою.

Виставлені ваги метрик дозволяють робити вибір конкретного стану під час вимірювання на користь того, який має найбільшу суму ваг. Наприклад, якщо для визначення рівня активності вимірювані метрики FDT вказують на стан «Active», а решта на стан «Sleep», тоді вибір буде на користь «Sleep», бо сума ваг для цього стану рівна 1,5 і більша за вагу FDT. Цей приклад може описувати випадок, коли користувач активував екран телефону і пристрій відобразив певний застосунок, однак обчислень не відбулось і можливо, що в наступний момент часу користувач перестане використовувати телефон. Якщо користувач продовжить його використовувати, то використання ресурсів систем виросте, що буде відображено у вимірюваних метриках.

Для визначення поточного рівня старіння необхідно обчислити середнє значення для N останніх записів часового ряду метрик M_{Aging} і визначити в який із інтервалів таблиці 1 вони входять. Для визначення поточного статусу батареї і активності використання необхідно отримати актуальні значення метрик на даний момент. Таким чином буде сформований вектор $P(0)$ і використаний для обчислення формули (2.2).

3.4. Метод омолодження програмного забезпечення для операційної системи Android

Запропоновані в другому розділі моделі та симуляції разом з розглянутими метриками та факторами старіння в попередніх підрозділах дають можливість розробити метод омолодження, який буде враховувати поступові рівні старіння, різні стратегії та механізми омолодження, активність використання мобільного пристрою та рівень заряду батареї. В даній роботі пропонується метод омолодження ПЗ, який дозволяє покращити як досвід користувача та продуктивність ПЗ, так і характеристики надійності. Загальну схему методу омолодження представлено у вигляді UML діаграми діяльності на рис. 3.17.

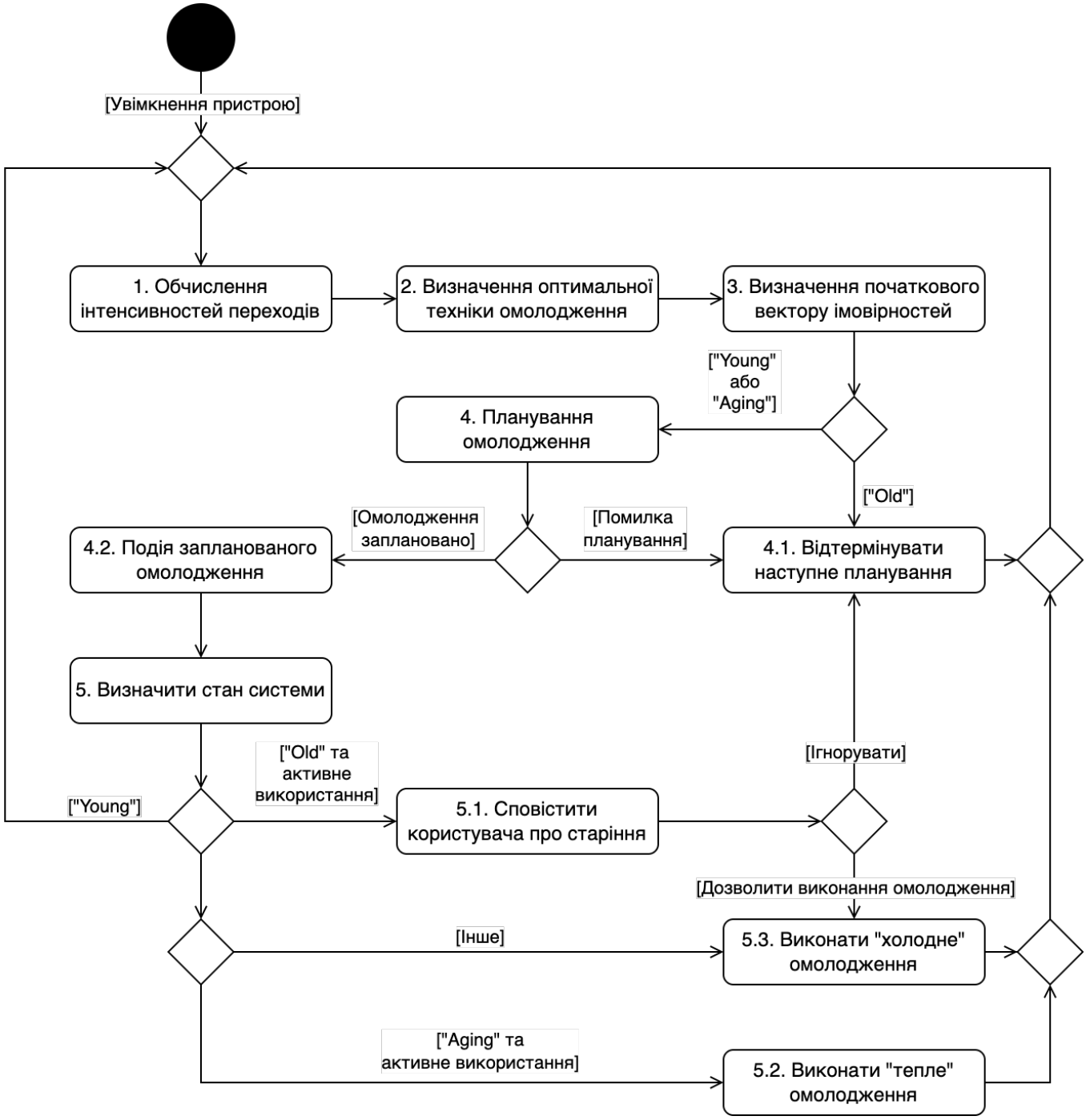


Рис. 3.17. UML-діаграма діяльності методу омолодження

На першому кроці методу омолодження необхідно виконати ініціалізацію інтенсивностей переходів, необхідних для моделювання процесу старіння. Цей крок використовує інтерфейс модуля моніторингу для обчислення матриці інтенсивності переходів Q.

Другий крок методу полягає у визначенні оптимальної для користувача техніки виконання омолодження під час прогнозування часу омолодження. Тобто, для комплексної моделі необхідно установити інтенсивність переходу між станами

«Active» та «Sleep» під час виконання омолодження таким чином, щоб при умові активного використання мобільного пристрою в пріоритеті було застосування «теплого» омолодження:

$a_{SmAmR} = a_{AmSmR} = 0$, якщо $a_{Am} > a_{Sm}$;

$a_{SmAmR} = a_{SmAm,}\ a_{AmSmR} = a_{AmSm}$, якщо $a_{Am} \leq a_{Sm}$.

На третьому кроці виконується ініціалізація початкового вектору імовірностей $P(0)$ з використанням інтерфейсу модуля моніторингу. Цей вектор однозначно визначає стан в якому перебуває система в даний момент.

Виконання кроків 1-3 можливе або після увімкнення мобільного пристрою, або після завершення виконання запланованої процедури омолодження. Ці дві умови означають, що на кроці 3 очікується перебування системи в стані «Young» або «Aging». Якщо система все ж таки перебуває в стані «Old», тоді виконання, так само як і планування, омолодження недоцільне, тому відбувається перехід до кроку 4.1. Така ситуація можлива в умовах, коли мобільний пристрій постійно перебуває в стані «Old» через помилки, не пов’язані із явищем старіння.

Якщо система перебуває в одному із станів «Young» чи «Old», тоді виконується крок 4, де відбувається пошук часу виконання омолодження з урахуванням сформульованих умов (2.7) комплексної моделі та вхідних даних, отриманих на кроках 1-3. Для прогнозованого часу омолодження необхідно передбачити порогові значення, вихід за які означає недостовірність отриманого результату, зокрема через те, що прогнозування залежить від якості статистичних даних та способу їх отримання.

Якщо прогнозований час омолодження $T_R < T_{RBottom}$, або $T_R > T_{RTop}$, то наступним потрібно виконувати крок 4.1, який полягає в очікуванні наступної спроби прогнозування омолодження певний час $T_{Postpone}$. Наприклад, якщо $T_{RBottom}$ = 30 хвилин, а T_R = 16 хвилин, тоді омолодження може бути надлишковим і мати більш негативний ефект. В той самий час, якщо T_{RTop} = 7 днів, а T_R = 10 днів, тоді є ймовірність пропустити реальний момент переходу в стан «Old» і саму відмову старіння. Визначення коректних і ефективних значень $T_{RBottom}$, T_{RTop} та $T_{Postpone}$

потребує додаткових досліджень, зокрема ці значення можуть бути адаптовані під конкретні умови середовища.

Якщо прогнозований час омолодження коректний, тоді виконується крок 4.2, тобто очікується запланована подія початку виконання процедури омолодження. Ця подія ініціалізує виконання п’ятого кроку, який використовує інтерфейс модуля моніторингу старіння і визначає стан системи для прийняття рішення про доцільність виконання процедури омолодження.

Якщо система перебуває в стані «Young», тоді виконання омолодження недоцільне і виконується повернення до першого кроку і повторного планування наступної процедури омолодження. Якщо система перебуває в стані «Old» і користувач активно використовує мобільний пристрій, то процедура «теплого» омолодження буде неефективна, а «холодне» омолодження може пошкодити користувачу, тому пропонується в цей момент показати повідомлення з інструкцією про необхідність перезавантажити мобільний пристрій, тобто виконати «холодне» омолодження. Якщо система перебуває в стані «Aging» і користувач активно використовує мобільний пристрій, то застосовується процедура «теплого» омолодження. В усіх інших випадках виконується «холодне» омолодження.

Використання «теплого» омолодження на ранньому етапі дозволяє уникнути переходу в стан «Old» та необхідності перезавантаження мобільної системи, тим самим покращується як UX, так і характеристики надійності.

Після виконання омолодження будь якого типу, метод передбачає, що система повернеться до стану «Young», тому важливим завданням є розроблення ефективних і коректних реалізацій методів «холодного», а особливо «теплого» омолодження.

В майбутніх роботах важливо виконати реалізацію запропонованого методу для ОС Android, перевірити запропоновані стратегії омолодження в реальних умовах та оцінити їх вплив на покращення метрик старіння. Зокрема, важливо перевірити та уточнити такі етапи методу як прогнозування часу омолодження, конкретні реалізації механізмів «теплого» та «холодного» омолодження, визначення стану системи на основі запропонованих метрик та їх порогів, інтервалів між вимірюваннями метрик, критеріїв $T_{RBottom}$, T_{RTop} та $T_{Postpone}$.

3.5. Висновки до розділу 3

В даному розділі представлено результати експериментальних досліджень явища старіння ПЗ в ОС Android, зокрема, метрики та фактори старіння.

Запропоновано та експериментально перевірено дві нові метрики GUI, в контексті старіння ПЗ: FDT та JFC. Метрика FDT може бути використана у більшій кількості реальних сценаріїв використання ПЗ, ніж метрика ALT, і дозволяє отримувати неперервні часові ряди даних. Наприклад, у виконаних експериментах метрика FDT дозволяє отримати в 4 рази більше записів, ніж метрика ALT. Крім того, метрика FDT дозволяє визначати інтенсивність використання пристрою користувачем в різні моменти часу.

Виконано аналіз оцінки процесів «oom_adj_score» з точки зору старіння ПЗ. Розглянута оцінка може бути застосована для визначення стану використання системи користувачем в методах прогнозування старіння та виконання процедури омолодження. Визначено дві умовні групи процесів за «oom_adj_score», а саме, групи системних та користувацьких процесів.

Визначено, що системні процеси system_server, com.android.systemui та surfaceflinger можуть бути використані в методах омолодження ПЗ в якості індикаторів старіння в ОС. Такі процеси як cameraserver, audioserver та інші можуть бути індикаторами в залежності від сценарію використання мобільного пристрою користувачем.

Користувацькі застосунки також вразливі до явища старіння, однак виконання процедури омолодження може бути недоцільним в контексті надійності ПЗ. Тому, важливо вживати заходів для попередження виникнення основних помилок старіння: попереджати помилки переповнення пам’яті; мінімізувати навантаження на основний потік UI.

Досліджено нові фактори старіння, а саме, крос-платформові застосунки на базі фреймворку Flutter та сценарій використання мобільного пристрою із паузами під час генерування робочого навантаження.

Результати показують, що крос-платформові застосунки Flutter можуть бути більш вразливими до ефектів старіння в контексті продуктивності UI, ніж стандартні

native застосунки. Як для Flutter застосунків, так і для native застосунків тренд збільшення метрики FDT спостерігається у 2 із 6 виконаних експериментів, однак, абсолютні значення метрики відображення кадрів UI для Flutter значно перевищують native – приблизно вдвічі.

Сценарій використання із паузами характерний тим, що процес старіння може продовжуватись навіть без взаємодії користувача із UI. Зокрема, в системному процесі system_server продовжує спостерігатись позитивний тренд збільшення метрики PSS, що говорить про необоротність процесів старіння в ОС та необхідність застосування омолодження саме в момент пауз активності для покращення досвіду користувача та попередження виникнення відмов старіння.

Запропоновано і описано метод омолодження ПЗ для ОС Android, який використовує комплексну модель старіння та омолодження на основі СТМС для прогнозування часу виконання процедури омолодження. Метод, так само як і модель, визначає поступові рівні старіння системи, що дозволяє будувати стратегії омолодження, які враховують актуальний стан старіння системи та попереджати перехід системи в стан з високою ймовірністю відмови старіння. Крім того, запропонований метод прогнозування враховує такі чинники як стан заряду батареї та активність використання мобільного пристрою користувачем, що дозволяє покращити UX. Метод передбачає можливість застосування механізмів «холодного» та «теплого» омолодження для різних умов старіння.

РОЗДІЛ 4. ПРОГРАМНІ ЗАСОБИ ДЛЯ ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ ЯВИЩА СТАРІННЯ ТА ОМОЛОДЖЕННЯ В ОПЕРАЦІЙНІЙ СИСТЕМІ ANDROID

4.1. Методологія експериментальних досліджень явища старіння програмного забезпечення в операційній системі Android

Дослідження процесів старіння в конкретних платформах та середовищах дозволяє визначити особливості впливу різних чинників на старіння, ефективні метрики виявлення старіння та вразливі до старіння компоненти системи. Для ОС Android розроблено методології та підходи виявлення та аналізу процесу старіння ПЗ [89, 91, 94, 102], в яких можна виділити шість основних кроків проведення дослідження. Загальну схему методології дослідження явища старіння ПЗ представлено на рис. 4.1. Основним підходом методології є виконання стресового тестування для симуляції процесу старіння з метою подальшого аналізу отриманих вимірювань метрик старіння.

Рис. 4.1. Загальна схема методології дослідження явища старіння ПЗ в ОС Android

Перший етап полягає у визначенні стратегії моніторингу. Метрики можуть прямо чи опосередковано вказувати на наявність старіння в системі. Наприклад,

збільшення часу відгуку інтерфейсу користувача вказує на погіршення продуктивності. Водночас, збільшення використання оперативної пам’яті чи дискового простору не обов’язково означатиме погіршення продуктивності, однак може спричинити відмову внаслідок нестачі пам’яті. Таким чином, важливим завданням на цьому кроці є вибір метрик та індикаторів системи, які будуть вимірюватись під час виконання експерименту. Крім метрик та індикаторів важливо визначити набори компонентів системи та застосунків користувача, які також будуть відстежуватись згідно певного набору метрик. Для виконання вимірювань реалізуються технічні засоби, що полягають у виборі та налаштуванні утиліт командного рядка, визначенні форматів даних та методів їх перетворення, пошуку даних в журналах системи про використання ресурсів, тощо.

Другий етап визначення алгоритму генерації робочого навантаження полягає в реалізації стресового тестування та імітації тривалого використання мобільного пристрою. На цьому етапі важливо передбачити можливість зміни інтенсивності генерування робочого навантаження, а також, налаштування подій введення різного типу. Таким чином, завданням алгоритму генерації робочого навантаження є можливість реалізації різних сценаріїв використання мобільного пристрою без необхідності виконання збору даних в реальних умовах.

Далі, ґрунтуючись на обраних стратегіях моніторингу та алгоритмі генерування робочого навантаження, розробляється план виконання стресового тестування. Вибір чинників, що можуть впливати на процес старіння, є важливим аспектом під час розроблення дослідження та планування експериментів. Чинники, як і вимірювані індикатори, можуть мати прямий чи опосередкований вплив на прояви ефектів старіння. Наприклад, конфігурація пристрою визначає обмеження ресурсів пам’яті та CPU, що може мати безпосередній вплив на погіршення продуктивності, а робоче навантаження в різних випадках може по різному активувати процес старіння. Під час побудови плану тестування враховуються різні комбінації факторів, забезпечуючи оптимальний час виконання тесту, який дозволить відтворити процес старіння в прийнятний для дослідження час.

Четвертий етап виконання стресових тестів полягає у автоматизації виконання попередніх трьох етапів в певному програмно-апаратному середовищі. Таким середовищем може бути ПК та мобільний пристрій, які з’єднані USB кабелем по якому відбувається їх комунікація. Зокрема, важливими аспектами є необхідність забезпечення стабільності, неперервності та розмежування виконання окремих тестових випадків. Наприклад, важливе стабільне з’єднання мобільного пристрою із пристроєм, який генерує робоче навантаження та акумулює метрики старіння, оскільки необхідно забезпечувати неперервне виконання одного тестового запуску. Переривання виконання тесту, якщо це не передбачено дизайном експерименту, може негативно вплинути на якість отриманих даних, оскільки в цьому випадку можлива втрата даних для формування повного часового ряду вимірюваних метрик, та втрата контролю над робочим навантаженням. Крім того, важливо розмежувати виконання окремих тестових запусків, якщо виконання тестів відбувається на одному і тому ж пристрої. Наприклад, необхідно виконати перезавантаження ОС, щоб позбавитись можливих наслідків накопичення помилок старіння.

Результатами виконання четвертого кроку є «сирі» дані у вигляді текстових документів чи таблиць, які містять записи із системних журналів чи файлової системи про використання пам’яті, виконання процесів, повідомлень про помилки та інші системні події. Дані в такому вигляді можуть бути незручні для виконання аналізу, тому важливим завданням є підготовка вхідних даних для п’ятого кроку, що зазвичай означає формування часових рядів із перетворення значень змінних у зручний для обробки формат.

На п’ятому етапі методології отримані часові ряди для вимірюваних метрик аналізуються з допомогою методів статистики, регресійного аналізу, машинного навчання, а також, методів аналізу часових рядів. Статистичні методи дозволяють визначити, які фактори посилюють погіршення продуктивності та споживання ресурсів. Встановлення взаємозв'язків між старінням ПЗ та показниками використання ресурсів дозволяє визначити, які підсистеми впливають на старіння, що забезпечує можливість розроблення стратегій омолодження ПЗ.

4.2. Розроблення середовища експериментальних досліджень явища старіння та ефективності метрик старіння в операційній системі Android

В межах даної роботи розроблено фреймворк, який реалізує загальну методологію із шести етапів для виконання експериментальних досліджень процесу старіння ПЗ, ефективності метрик старіння та впливу факторів старіння. Фреймворк складається із трьох модулів:

- Модуль виконання стрес тестів та вимірювання метрик старіння;
- Модуль перетворення системних метрик в часові ряди;
- Модуль аналізу часових рядів.

Модуль виконання стрес тестів та вимірювання метрик реалізує та автоматизує перші чотири етапи методології. Середовищем (рис. 4.2), яке використано для реалізації цього модуля та виконання тестів, є ПК із ОС Windows 10 та мобільні пристрої різних моделей із ОС Android. З'єднання пристроїв відбувалось через USB інтерфейс та утиліту командного рядка adb [96] для обміну командами та збирання даних з мобільного пристрою. Програмним середовищем для виконання команд є Windows Power Shall [114].

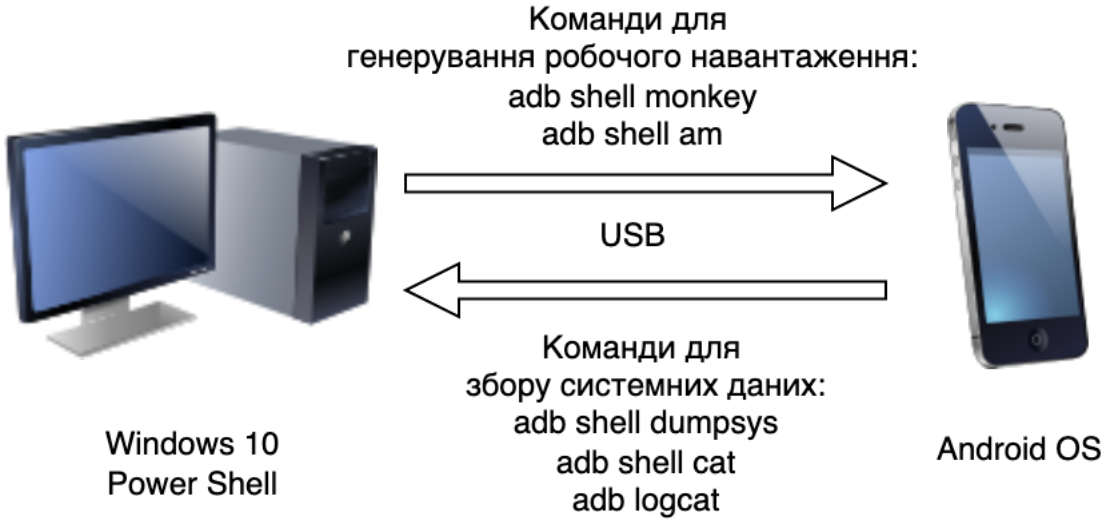


Рис. 4.2. Програмно-апаратне середовище виконання стресових тестів

Повна імплементація основних методів збору системних даних та генерування робочого навантаження подана в додатку В.

Для отримання даних про використання системних ресурсів використано утиліту Android dumpsys [107], що виконується на мобільному пристрої та дозволяє зібрати інформацію про системні сервіси. Агрегована статистика про відображення кадрів для кожного застосунку отримується з допомогою команди gfxinfo та вказаної назви пакета цього застосунку в якості параметра. Для отримання детальних даних про час і тривалість відображення кадрів використовувалась команда framestats.

Функція `DumpsysServiceMonitor` дозволяє налаштувати виконання команди dumpsys для різних сервісів (`$Service`) із необов'язковими параметрами (`$OptionalParams`), яка може бути виконана для списку користувацьких застосунків (`$Packages`):

```
function DumpsysServiceMonitor
{
    Param(
        [String] $Output,
        [String] $Service,
        [String[]] $Packages,
        [String] $OptionalParams = ""
    )
    ...
        adb shell dumpsys $Service $OptionalParams | Out-File -FilePath $fullPath -
Append
    ...
}
```

Приклад повної команди для збору інформації про кадри застосунку інтернет браузера:

```
adb shell dumpsys gfxinfo com.android.chrome framestats
```

Ця команда повертає багато даних, серед яких необхідно виділити 4 важливих записи, необхідні для цього дослідження: Total Frames Rendered, Janky Frames, INTENDED_VSYNC та FRAME_COMPLETED.

Системний журнал повідомлень Logcat [115] включає записи системних сервісів, таких як ActivityManager та ART. Рядки записів ART містять дані про тривалість затримки та загальну тривалість виконання задачі збирача сміття в певному процесі. А ActivityManager генерує інформаційні повідомлення про завершення виконання відображення Android Activity.

Функція LogCatMonitor дозволяє вивести записи системного журналу LogCat в текстовий файл на ПК.

```
function LogCatMonitor
{
    Param([String] $Output)
    ...
    adb logcat -d -v monotonic | Out-File -FilePath $path -Append
    adb logcat -c
}
```

Ще одним джерелом даних є файлова система proc ядра Linux, яка забезпечує доступ до файлів, які містять інформацію про використання пам'яті, дискового простору, планування та пам'ять процесів. Функція ProcTasksMonitor дозволяє вивести записи виконання процесів системи в текстовий файл на ПК:

```
function ProcTasksMonitor {
    Param([String] $Output)
    ...
    $dirs_str = adb shell ls /proc/
    ...
    $dirs_str_full = $dirs_str | ForEach-Object { "/proc/" + $_ + "/stat" }
    ...
    $res = adb shell cat ($dirs_str_full -join ' ')
    ...
}
```

Ця функція виконує читання піддиректорій із директорії /proc/ для визначення місцезнаходження даних статистики окремих процесів. Після чого, для кожної знайденої директорії виконується прочитання файлу /stat, дані якого записуються в один масив даних, який пізніше записується в текстовий файл.

Для генерування робочого навантаження використовувався застосунок командного рядка monkey [95], який виконується на емуляторі чи мобільному пристрої і генерує псевдовипадковий потік таких подій як натискання на кнопки, доторкання до дисплею, жести, а також різні події рівня системи. Функція RunMonkey дозволяє налаштувати виконання команди:

```
function RunMonkey
{
    param (
        [String[]] $Packages,
```

```
        [Int32] $DurationMs = 0,
        [Int32] $EventsCount = 100,
        [Boolean] $IgnoreErrors = $true,
        [Boolean] $EnableEvents = $true,
        [Boolean] $EnableSwitches = $true
    )
    ...
    adb shell monkey $packagesParams -v -v $throttleParam $eventsParams $ignoreParams
$EventsCount
}
```

Приклад команди monkey:

```
adb shell monkey [options] <event-count>
```

Список пакетів програм `$Packages` визначає ті застосунки користувача для яких буде виконуватись генерування подій введення та перемикання між цими застосунками. Параметр `$DurationMs` визначає тривалість генерування робочого навантаження, а `$EventsCount` визначає кількість псевдовипадкових подій. Параметри `$EnableEvents` та `$EnableSwitches` дозволяють увімкнути та вимкнути генерування подій введення та перемикання між застосунками. Не менш важливим є параметр `$IgnoreErrors`, який у всіх тестових випадках виконуваних у межах даної роботи є увімкненим, оскільки він дозволяє не переривати роботу стресового тестування у разі виникнення виняткової ситуацій, яка б привела до збою в роботі окремого застосунку.

Для реалізації тестових випадків із перезавантаженнями застосунків використовувався засіб командного рядка Android am (Activity Manager) та його команда `force-stop`. Виконання цих команд реалізовано з допомогою функцій `RunPackages` та `KillPackages`, які для списку пакетів користувацьких застосунків почергово виконують виконання відповідних команд:

```
foreach ($package in $Packages)
{
    adb shell monkey -p $package -c android.intent.category.LAUNCHER 1
}
foreach ($package in $Packages)
{
    adb shell am force-stop $package
}
```

Основною функцією входу у виконання тестового випадку є RunStressTest:

```
function RunStressTest {
    param (
        [Int32] $IntervalsCount,
        [String] $Cleaner,
        [String] $WorkloadGenerator,
        [String] $Monitor,
        [Int32] $WorkloadDurationTicks = 0,
        [Int32] $PauseDurationSeconds = 0,
        [String] $OnTimeAction = "",
        [Int32] $ActionTime = 0
    )
    ...
}
```

Параметр $IntervalsCount визначає кількість запусків функцій генерації робочого навантаження $WorkloadGenerator та збору системних даних $Monitor. Тривалість виконання окремих інтервалів генерування робочого навантаження визначається безпосередньо в функції $WorkloadGenerator, наприклад, через параметри функції RunMonkey. Параметр $WorkloadDurationTicks визначає кількість послідовно виконуваних генерацій робочого навантаження та збору даних до настання паузи, тривалість якої визначається параметром $PauseDurationSeconds. Чергування робочого навантаження і пауз триває до того часу, поки не виконається вся кількість $IntervalsCount.

Загальна тривалість експерименту може бути визначена добутком значення $IntervalsCount та середнього часу, необхідного на виконання робочого навантаження та збору системних даних. Якщо експеримент передбачає сценарій, де робоче навантаження не виконується, тобто, значення $PauseDurationSeconds більше нуля, то загальна тривалість тестового випадку збільшиться на відповідний час перебування алгоритму навантаження на паузі.

Параметри $OnTimeAction та $ActionTime допомагають реалізувати додаткові випадки та виконання подій у визначений час. Наприклад, для виконання тестових випадків експериментального дослідження EXP2 (див. додаток Б) ці

параметри було використано для виконання перезавантаження усіх застосунків приблизно в середині часу виконання тесту. Така необхідність виникла внаслідок того, що кожен експеримент тривав понад чотири години, тому в деяких застосунках в певний момент спостерігались так би мовити «тупикові» ситуації, коли псевдовипадкові події введення не забезпечували вихід із деяких областей програми. Тому, перезавантаження застосунків дозволило виправити цей недолік без значного впливу на остаточний результат дослідження.

Текстові файли frames_draw_and_launch_time_comparison_test.ps1 та native_cross_comparison_test.ps1, які подані в додатку В, повністю описують програмний код для виконання тестових сценаріїв експериментальних досліджень EXP1 та EXP2 відповідно.

Модуль перетворення системних метрик в часові ряди забезпечує інструментами, які дозволяють сформувати часові ряди із «сирих» даних, отриманих під час виконання стресових тестів. Основними завданнями цього модуля є читання текстових файлів, перетворення вимірюваних значень метрик в зручний для оброблення формат, синхронізація значень окремих змінних в один запис часового ряду.

Програмна реалізація цього модуля виконана з допомогою мови Kotlin в середовищі розробки IntelliJ Idea. Використання ООП під час розроблення модуля дозволяє розширювати функціональні можливості, зокрема, шляхом додавання нових типів вхідних даних та вихідних часових рядів. Ознайомитись із повною реалізацією даного модуля можна за посиланням на відкритий репозиторій [116]. Програмна реалізація модуля формування часових рядів складається із трьох підпакетів:

- Core – програмний код, який реалізує основні функціональні можливості модуля та абстракції, необхідні для забезпечення можливостей розширення;
- Datasets – конкретні реалізації класів та методів читання файлів даних, перетворення даних в моделі та об’єднання в часові ряди;
- Utils – класи та функції, які реалізовують такі загальні функції, як читання файлів та запис таблиць із часовими рядами у форматі даних .csv.

Для конвертування даних, зібраних із різних ресурсів у текстових файлах, розроблено класи `MultiLineLogsParser` та `SingleLineLogsParser`, які реалізують метод `parse()` інтерфейсу `LogsParser`. Клас `MultiLineLogsParser` реалізує алгоритм оброблення текстового файлу, де окремі записи, які потрібно конвертувати в модель даних, можуть бути багаторядковими, а розрізнити ці записи можна з допомогою виокремлення першого та останнього рядків із особливим форматом. В свою чергу, текстові документи, де кожен рядок є унікальним записом, можуть бути конвертовані в набір моделей даних з допомогою класу `SingleLineLogsParser`. Класи `MultiLineRecordConverter` та `SingleLineRecordConverter` призначенні для конвертування окремих багаторядкових та однорядкових блоків даних, відповідно.

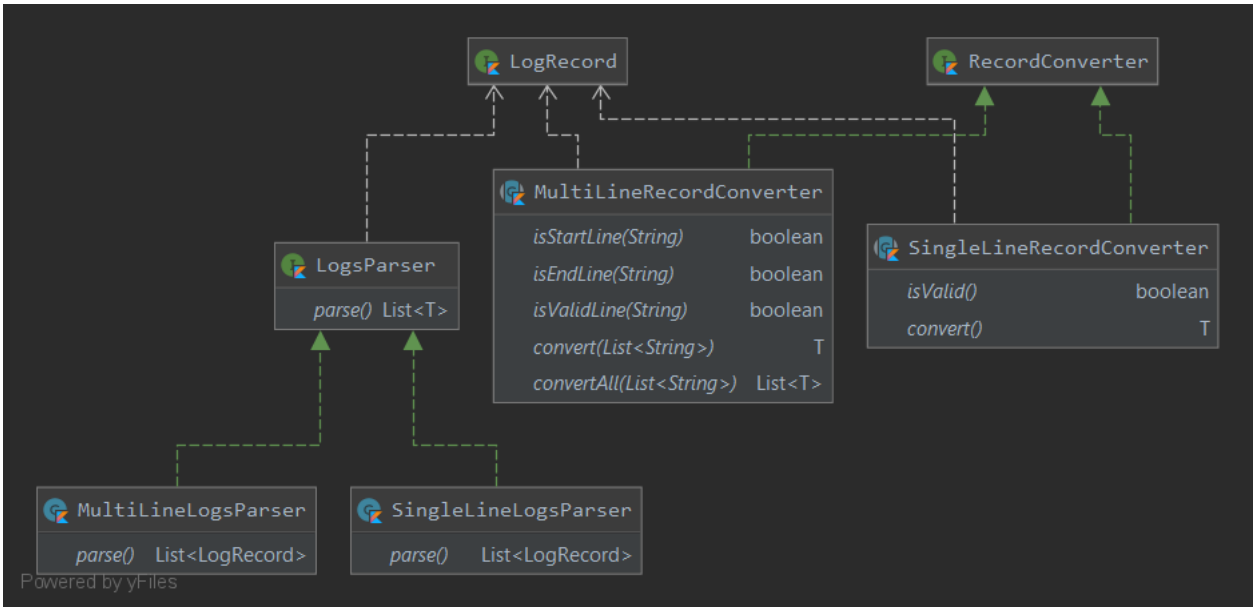


Рис. 4.3. UML-діаграма класів для конвертування даних із текстових файлів у моделі

Існують різні джерела вхідних даних, які необхідно об’єднати в один часовий ряд. Наприклад, під час виконання стресового тесту кожні 60 секунд виконувався збір даних використання оперативної пам’яті та статистики відображення кадрів GUI. Системні дані по кожній із цих метрик мають свої часові мітки, тому важливим завданням є часова синхронізація записів зібраних даних. Відповідно, для формування часових рядів на основі даних моделей, які отримано після

конвертування текстових файлів із «сирими» даними, розроблено класи `TimeSeriesGenerator`, `IntermediateRecord` та `TimeSeriesRecord`.

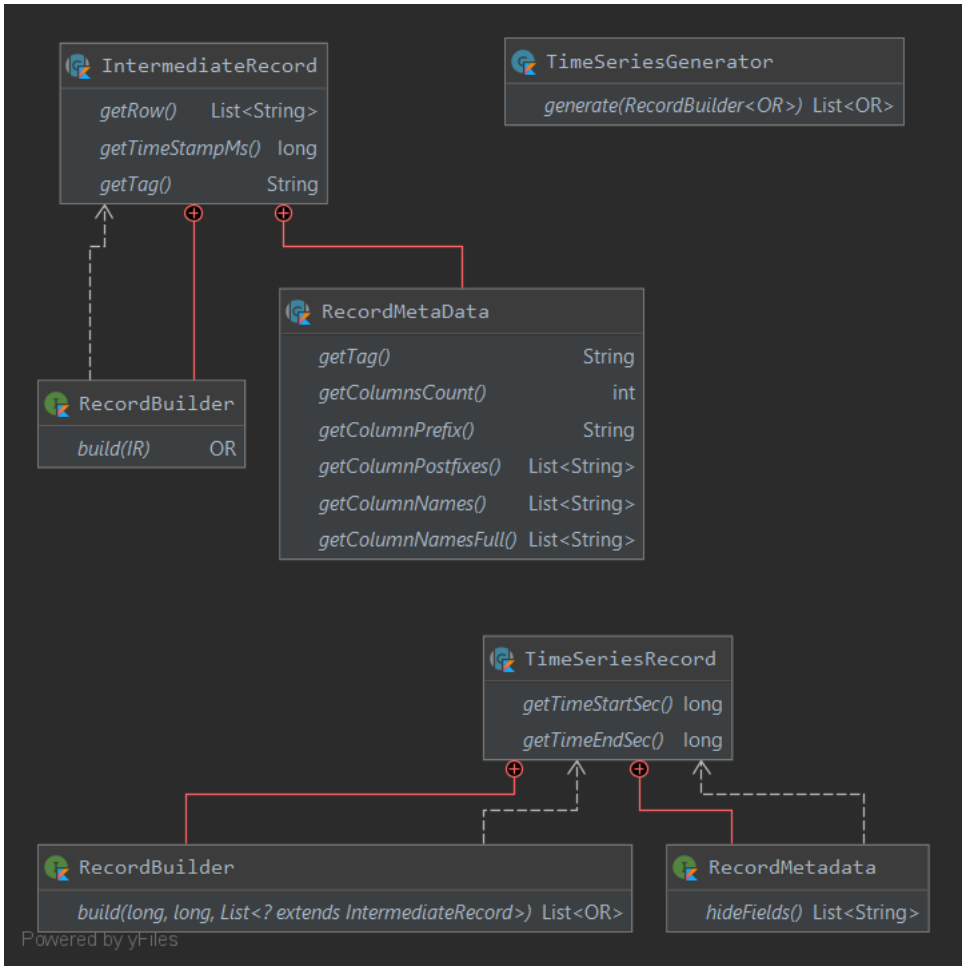


Рис. 4.4. UML-діаграма класів для представлення та формування часових рядів

В методі generate класу `TimeSeriesGenerator` реалізовано основну функцію формування часового ряду із встановленим інтервалом часу `intervalSec` для групування записів проміжних моделей даних `IntermediateRecord` в один запис `TimeSeriesRecord`.

`IntermediateRecord` представляє проміжну модель даних, яка призначена для перетворення запису `LogRecord` в необхідний формат часового ряду. З допомогою внутрішнього абстрактного класу `RecordMetaData` проміжна модель описується набором метаданих, які визначають назви полів даних, що будуть використовуватись в остаточній таблиці. Внутрішній інтерфейс `RecordBuilder`,

який описує метод `build()`, призначений безпосередньо для перетворення вхідної моделі даних у об'єкт типу `IntermediateRecord`.

Абстрактна модель `TimeSeriesRecord` – це запис часового ряду, який в батьківських класах може включати набір записів типу `IntermediateRecord`, а також, описує інтервал часу в який ці записи входять. При цьому, метод `build` внутрішнього інтерфейс класу `RecordBuilder` призначений для групування та агрегування списку записів `IntermediateRecord` по їх типу, даних та часових мітках. В результаті роботи цього методу отримуємо унікальний запис часового ряду для певного інтервалу часу, який містить агреговані записи різного типу, успадковані від `IntermediateRecord`. Наприклад, якщо протягом 60-ти секунд генерування робочого навантаження відбувалось перемикання між п'ятьма різними застосунками, то очікується, що буде п'ять записів `IntermediateRecord`, які описують статистику відображення кадрів GUI для кожного застосунку. Для формування одного запису часового ряду необхідно агрегувати зібрані дані, зокрема, застосовуючи середнє арифметичне чи визначаючи медіану цих записів. Внутрішній інтерфейс класу `RecordMetadata` та його метод `hideFields()` дозволяє приховати поля даних, які потрібно виключити із результуючої таблиці, наприклад, це може бути використано для випадків, коли окремі записи `IntermediateRecord` можуть містити однакові по змісту поля даних.

Наступні класи конвертерів реалізують інтерфейс базових класів `MultiLineLogsParser` та `SingleLineLogsParser` для читання текстових файлів та генерування масивів моделей даних, що наслідують базовий тип `LogRecord`.

Клас `LCActivityRecordConverter` конвертує один рядок журналу LogCat в модель даних `LCActivityRecord`, яка містить інформацію про подію відображення екрану застосунку Android, яка генерується сервісом ActivityManager. Приклад рядка, який містить відповідний запис:

```
1301331.400 1604 1625 I ActivityManager: Displayed
com.google.android.calendar/.launch.oobe.WhatsNewFullScreen: +1s26ms (total +1s216ms)
```

Клас LCGarbageCollectorRecord реалізує конвертування рядка журналу LogCat в модель даних LCGarbageCollectorRecord, який містить інформацію про подію затримки в роботі збирача сміття. Подія генерується системним сервісом art. Приклад рядка, який містить відповідний запис:

```
132.355  1665  1674 I art      : Background partial concurrent mark sweep GC freed 56110(3MB) AllocSpace objects, 37(1452KB) LOS objects, 33% free, 32MB/48MB, paused 2.088ms total 125.289ms
```

В обох випадках, записи містять спільну інформацію, а саме, мітку часу в наносекундах із моменту ввімкнення мобільного пристрою, ідентифікатори процесу та потоку, який згенерував відповідне повідомлення, мітку пріоритету даного повідомлення, а також назву сервісу. Після назви сервісу і символу «:» слідує специфічна для кожної події інформація.

Для опрацювання вихідних даних команди adb shell dumpsys gfxinfo реалізовано два конвертери багаторядкових блоків текстового файлу та відповідні їм моделі даних, а саме, GfxInfoFrameStatsRecordConverter (GfxInfoFrameStatsRecord) та GfxInfoFrameTimingRecordConverter (GfxInfoFrameTimingRecord).

Модель GfxInfoFrameStatsRecord описує статистичні дані про відображення кадрів GUI для певного користувацького застосунку, зокрема такі, що показують кількість пропущених кадрів та причини затримки відображення цих кадрів:

```
** Graphics info for pid 4930 [com.android.chrome] **
Stats since: 78871867312ns
Total frames rendered: 8
Janky frames: 6 (75.00%)
50th percentile: 25ms
90th percentile: 250ms
95th percentile: 250ms
99th percentile: 250ms
Number Missed Vsync: 3
Number High input latency: 0
Number Slow UI thread: 4
Number Slow bitmap uploads: 0
Number Slow issue draw commands: 4
```

```
HISTOGRAM: 5ms=0 6ms=2 7ms=0 …
```

В свою чергу, `GfxInfoFrameTimingRecord` відображає дані, які описують часові мітки подій відображення кожного кадру інтерфейсу:

```
---PROFILEDATA---
Flags,IntendedVsync,Vsync,OldestInputEvent,NewestInputEvent,HandleInputStart,Animatio
nStart,PerformTraversalsStart,DrawStart,SyncQueued,SyncStart,IssueDrawCommandsStart,S
wapBuffers,FrameCompleted,DequeueBufferDuration,QueueBufferDuration,
1,82153941701,82303941695,9223372036854775807,0,82309223352,82309259967,82309263040,8
2316047831,82356475748,82356603248,82358091477,82429673300,82433628300,9742000,300000
,
...
---PROFILEDATA---
```

Для отримання даних про використання пам’яті процесами використовується команда `adb shell dumpsys`, яка генерує наступну відповідь, яка конвертується в модель `MemInfoProcRecord` з допомогою `MemInfoProcRecordConverter`:

```
time,144070,144070
…
proc,pers,com.android.systemui,1737,80867,N/A,e
proc,pers,system,1520,177296,N/A,e
…
ram,2917296,1626742,1888381
lostram,-53157
…
```

Рядок time містить мітку часу в наносекундах з моменту запуску системи. Рядки proc містять необхідну інформацію про пам’ять процесів системи, зокрема, oom_adj_score (пріоритетність процесу), назву та ідентифікатор процесу, значення PSS. Крім того, виокремлюються також дані про загальний об’єм зайнятої та вільної пам’яті, а також, об’єм «загубленої» пам’яті.

Клас конвертер `ProcTaskRecordConverter` та модель `ProcTaskRecord` дозволяють описати дані використання процесорного часу процесами системи, а також помилки читання сторінок пам’яті, які отримані з файлової системи Android:

```
…
105 (edac-poller) S 2 0 0 0 -1 69238880 0 0 0 0 0 0 0 0 0 -20 1 0 44 0 0
18446744073709551615 0 0 0 0 0 0 0 2147483647 0 18446744073709551615 0 0 17 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
106 (system) S 2 0 0 0 -1 1075871808 0 0 0 0 0 11 0 0 20 0 1 0 57 0 0 18446744073709551615
0 0 0 0 0 0 0 2147483647 0 18446744073709551615 0 0 17 6 0 5 0 0 0 0 0 0 0 0 0 0 0 0
```

```
107 (ipa_power_mgmt) S 2 0 0 0 -1 69238880 0 0 0 0 0 0 0 0 0 -20 1 0 58 0 0
18446744073709551615 0 0 0 0 0 0 0 2147483647 0 18446744073709551615 0 0 17 4 0 0 0 0
0 0 0 0 0 0 0 0 0 0
…
```

Ці записи дають змогу отримати інформацію про стан процесу, кількість мінорних та мажорних помилок сторінок, тривалість часу в режимі користувача та ядра, кількість потоків та інше.

Проміжні моделі для представлення даних в часових рядах наступні:

- `FrameStatsRecord` – це модель, що описує такі метрики процесів та застосунків із графічним інтерфейсом як загальна кількість відображених кадрів, загальна кількість «зіпсованих» кадрів, кількість кадрів не відображених через певні причини, зокрема, через навантаження обчислень на потік UI, тривале виконання команд малювання чи обробки растрових зображень;

- `FrameTimeRecord` – це модель, яка описує метрику тривалості відображення кадру GUI, або ж метрику FDT, конкретного користувацького застосунку;

- `LaunchTimeRecord` – це модель, яка описує метрику тривалості відображення екрану Android застосунку, вона ж є метрикою ALT;

- `GarbageCollectorRecord` – це модель, яка описує такі метрики збирача сміття як Pause Time та Total Time;

- `RamUsageRecord` – це модель, що описує використовувану пам’ять процесом з допомогою метрики PSS, а також, включає такі метрики як загальна кількість використовуваної, вільної та загубленої оперативної пам’яті.

Описані вище моделі метрик можуть бути об’єднані в один із наступних часових рядів:

- `PackageMetricsRecord` – модель запису часового ряду, яка групує всі попередньо описані проміжні моделі по назвах процесів та користувацьких застосунків. Дозволяє проводити аналіз явища старіння як на рівні всієї системи, так і в окремих процесах;

- `UserPerceivedResponseMetricsRecord` – модель запису часового ряду, яка агрегує дані моделей `RamUsageRecord`, `FrameTimeRecord`,

`LaunchTimeRecord` та `FrameStatsRecord`. Цей часовий ряд не враховує групування по окремих процесах та користувацьких застосунках і призначений для аналізу метрик продуктивності користувацького інтерфейсу та використання пам'яті виключно на рівні всієї системи.

Модуль аналізу часових рядів забезпечує інструментами дослідження даних, отриманих із двох попередніх модулів. Середовище, в якому реалізовано модуль, є Google Colab [117], яке дозволяє писати та виконувати програмний код мовою Python прямо в браузері і широко застосовується для задач машинного навчання та аналізу даних. Colab являє собою сервіс Jupyter ноутбука, який не потребує ніяких налаштувань та є повністю безплатним з доступом до апаратно-обчислювальних ресурсів, зокрема, GPU.

Для результатів експериментальних досліджень EXP1 та EXP2 (див. додаток Б) розроблено три підмодуля, які призначені для оброблення та аналізу різних часових рядів з урахуванням різних аспектів досліджуваного явища:

- User-Perceived Response Metrics.ipynb – це підмодуль аналізу метрик продуктивності користувацького інтерфейсу в контексті процесу старіння ПЗ, зокрема, порівняння метрик ALT, FDT та JFC;
- Aging-Related Services.ipynb – це підмодуль аналізу системних процесів та користувацьких застосунків з точки зору їх ролі в процесі старіння та впливу старіння на них;
- Aging Factors - Flutter and Usage Delay.ipynb – це модуль аналізу впливу таких факторів як Flutter застосунки та сценарій використання із паузами генерування робочого навантаження.

Процес аналізу даних з допомогою розроблених підмодулів в тій чи іншій мірі враховує наступні етапи та техніки:

1) Визначення цілей дослідження;
2) Розуміння даних;
3) Підготовка даних;
4) Побудова моделей;
5) Представлення результатів.

Визначення цілей дослідження – це початковий етап, який передує написанню програмного коду. Виходячи із постановки завдання виконуваного дослідження та отриманих тестових даних у вигляді часових рядів різних метрик старіння та системних даних, що забезпечують контекстною інформацією, важливо визначити які методи для яких даних можуть бути застосовані. Наприклад, методи статистики можуть бути застосовані для порівняння даних однакових змінних, але отриманих із різних експериментів. Методи аналізу часових рядів можна використовувати для визначення трендів вимірюваних метрик, які, гіпотетично, можуть бути репрезентативними з точки зору процесу старіння.

Етап розуміння даних дозволяє попередньо оцінити природу тестових даних та виявити можливі помилки отримані під час їх генерування. В середовищі Colab для завантаження і маніпуляцій над даними використовується бібліотека `pandas`, яка дозволяє завантажити *.csv файл у вигляді об'єкта `DataFrame`. Метод `describe()` дозволяє відобразити узагальнену статистичну інформацію для всіх полів таблиці даних, зокрема, загальну кількість ненульових рядків, середнє, медіанне, мінімальне та максимальне значення. Перевірка цих значень дозволить завчасно виявити очевидні помилки в записах. Крім того, метод `info()` дозволяє відобразити кількість ненульових рядків для кожного поля та тип цього поля. Наприклад, якщо поле метрики FDT по тій чи іншій причині матиме тип відмінний від числового, то з таким полем неможливо проводити обчислення, тому це необхідно виправляти.

Ще один важливий аспект, який потребує перевірки – це наявність дублікатів в даних, які негативно впливатимуть на виконання аналізу. Тому важливо на ранньому етапі ідентифікувати можливі дублікати і виконати необхідні виправлення в алгоритмах модуля стресового тестування та генерування часових рядів, оскільки в більшості випадків регулярні дублікати можуть з'являтись саме через некоректний збір даних. Для перевірки наявності дублікатів використано методи `duplicated()` та `unique()`.

Крім перелічених вище методів, які дозволяють виявити очевидні помилки, на етапі розуміння даних також застосовуються методи виявлення кореляцій між змінними, типів розподілів даних та викидів (outliers).

Виявлені позитивні чи негативні кореляції дозволяють зробити припущення про можливі причинно-наслідкові зв'язки між двома випадковими змінними або відсутність цих взаємозв'язків. Наприклад, наступний метод дозволяє відобразити кореляційну матрицю (рис. 4.5) для вибраних даних:

```
dataForCorr =
data[['pss','pss_avg','launch_time','draw_time','janky_frames']]
corrMatrix = dataForCorr.corr(method="spearman")
sn.heatmap(corrMatrix, annot=True)
```

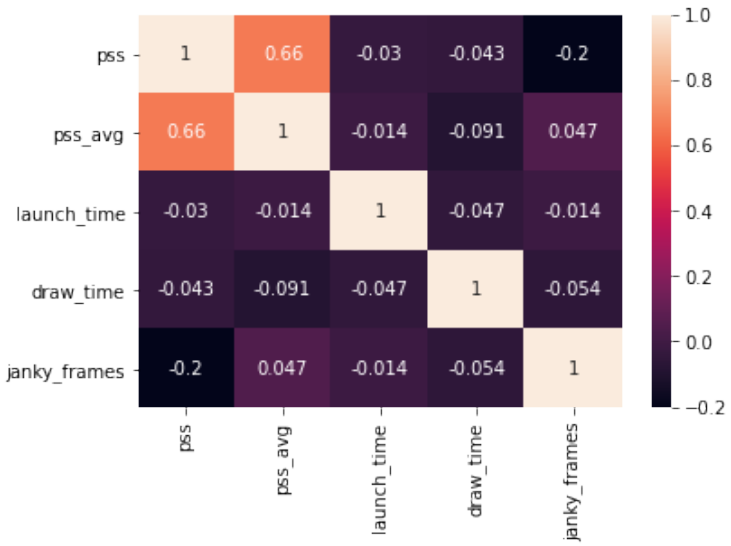


Рис. 4.5. Приклад кореляційної матриці для метрик PSS, ALT, FDT та JFC

Викиди являють собою результати вимірювань, які виділяються із загальної вибірки. Причинами їх появи можуть бути помилки вимірювань, особливості вимірюваного процесу, а також, частини розподілу. В даній роботі використано як математичні методи виявлення викидів, так і графічні. Математичними методами є розрахунок IQR (міжквартильний розмах) та z-оцінки. Графічними методами є представлення даних у вигляді точкової діаграми у випадку двох змінних та діаграми розмаху для випадку однієї змінної.

Наступний метод обчислення значення z дозволяє отримати список записів, які значно відхиляються від вибірки даних:

```
z = np.abs(stats.zscore(data[['draw_time']]))
print("draw_time outliers")
print(data[(z >= 3).all(axis=1)])
```

В свою чергу, команда `sn.boxplot(x=data.janky_frames)` дозволяє побудувати діаграму розмаху для обраної змінної. Результат виконання цієї команди (рис. 4.6) показує, що існує багато записів, що відрізняються від решти даних і один запис, який відрізняється навіть більше, ніж інші. Цей запис із значенням понад 400 може бути природнім, хоча і дуже рідкісним випадком, або ж помилковим, тому, для подальшої роботи з масивом даних, варто видалити такий запис.

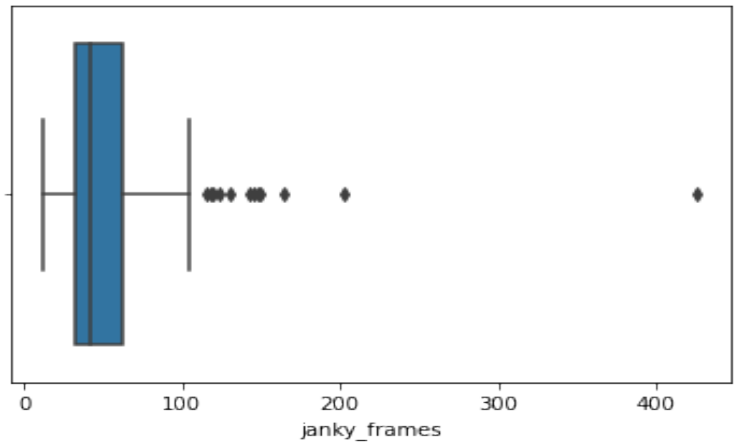


Рис. 4.6. Діаграма розмаху значень змінної JFC

Наступним етапом є підготовка даних до наступних процедур побудови регресійних моделей, аналізу часових рядів та візуалізацій. Цей крок передбачає виправлення знайдених помилок в даних, а саме, видалення дублікатів, виправлення типів даних та приведення до необхідних форматів. Крім того, на цьому етапі можна обчислити додаткові проміжні дані на основі вже відомих. Наприклад, обчислення відсоткового значення пропущених кадрів на основі даних про кількість всіх кадрів та кількість пропущених. Також на цьому кроці виконується видалення тих значень викидів, які можуть бути помилковими, або можуть мати негативний вплив на результати подальшого аналізу та моделювання. Додатково, можна виконати перейменування полів даних на такі, які будуть зрозумілі і читабельні в контексті задачі виконуваного дослідження.

В межах виконуваного дослідження побудовано прості моделі лінійної регресії на основі яких можна показати застосовність нових метрик відображення кадрів для

прогнозування часу до відмови. Наприклад, наступна програмна реалізація лінійної регресії виконана для метрики FDT:

```
Y = pd.DataFrame(data, columns = ['draw_time_ms'])
X = pd.DataFrame(data, columns = ['time_start_min'])
minutes = pd.DataFrame(filtered_data, columns = ['time_start_min'])
regressor = LinearRegression()
regressor.fit(X, Y)
model = regressor.predict(X)
```

Для виконання однофакторного дисперсійного аналізу застосовано метод `anova_lm` бібліотеки `statsmodels.api` для метрик FDT та JFR:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
lm = ols('draw_time ~ app',data=factor_analysis_data).fit()
table = sm.stats.anova_lm(lm)
print(table)
lm = ols('draw_time ~ usage',data=factor_analysis_data).fit()
table = sm.stats.anova_lm(lm)
print(table)
lm = ols('janky_ratio ~ app',data=factor_analysis_data).fit()
table = sm.stats.anova_lm(lm)
print(table)
lm = ols('janky_ratio ~ usage',data=factor_analysis_data).fit()
table = sm.stats.anova_lm(lm)
print(table)
```

Для аналізу системних процесів, які вразливі до ефектів старіння, виконано аналіз трендів збільшення значення метрик PSS, GC Duration Time та GC Pause Time. Тест Манна-Кандала використано для виявлення позитивного тренду в кожному тестовому випадку для кожної метрики окремо. Отримані результати зведено в окрему таблицю. Приклад програмної реалізації виявлення трендів старіння:

```
pss_trends = {}
gc_total_trends = {}
gc_paused_trends = {}
for file in processes_data:
  data = processes_data[file]
  data = data[data.process_group.isin(['native', 'pers'])]
  for process in data.package.unique():
    process_data = data[data.package == process]
    pss = process_data.pss
    if (pss.size > 1) & (pss.mean() > 20480):
```

```
      test = mk.original_test(pss, alpha=0.1)
      if test.h == True and test.slope > 0:
        if process in pss_trends:
          pss_trends[process] += 1
        else:
          pss_trends[process] = 1
    gc_paused_time = process_data.paused_time
    ...
    gc_total_time = process_data.total_time
    ...
memory_trends = {
    "pss" : pss_trends,
    "gc_duration" : gc_total_trends,
    "gc_paused" : gc_paused_trends
}
```

Для візуалізації отриманих результатів використано бібліотеки matplotlib та seaborn, які дозволяють будувати прості розподіли значень змінних з допомогою методу `distplot`, так і комплексні візуалізації, які складаються із різних шарів та елементів. Наприклад, рис. 3.1 отримано з допомогою наступної програмної реалізації:

```
fig, ax1 = plt.subplots()
fig.set_size_inches(8, 5)
plt.title('Launch Time vs Frame Draw Time Appearence')
plt.suptitle('Apps restart every 30 seconds', y = 0.92)
vis_data = data[data.time_start_min < 60]
vis_data.loc[vis_data['launch_time'] == 0, 'launch_time'] = np.nan
time_axis_data = vis_data.time_start_min
color = 'blue'
ax1.set_xlabel('Time (min)')
ax1.set_ylabel('Frame Time (ms)', color=color)
ax1.plot(time_axis_data, vis_data.draw_time_ms, color=color, linewidth=0.5)
ax1.tick_params(axis='y')
ax2 = ax1.twinx()
color = 'red'
ax2.set_ylabel('Launch Time (ms)', color=color)
ax2.scatter(time_axis_data, vis_data.launch_time, color=color, s=2)
ax2.vlines(time_axis_data,  pd.DataFrame(np.zeros((vis_data.launch_time.size,  1))),
vis_data.launch_time, color=color, linewidth=0.1, linestyles='dashed')
ax2.tick_params(axis='y')
fig.tight_layout()
```

```
plt.show()
```

4.3. Розроблення структури модулів програмного засобу омолодження програмного забезпечення для операційної системи Android

Для реалізації запропонованого в третьому розділі методу омолодження ПЗ в ОС Android необхідно передбачити комплекс програмних модулів, які б забезпечили як виконання методу та прогнозування оптимального часу омолодження, так і моніторинг процесу старіння в системі. На рис. 4.7 показано діаграму компонент сервісу старіння та омолодження для ОС Android, де виділено два модулі Aging Metrics Monitor та Rejuvenation Method. Крім цього, на діаграмі показано модуль, який представляє користувацький застосунок Android та способи його взаємодії через інтерфейси із модулями старіння та омолодження.

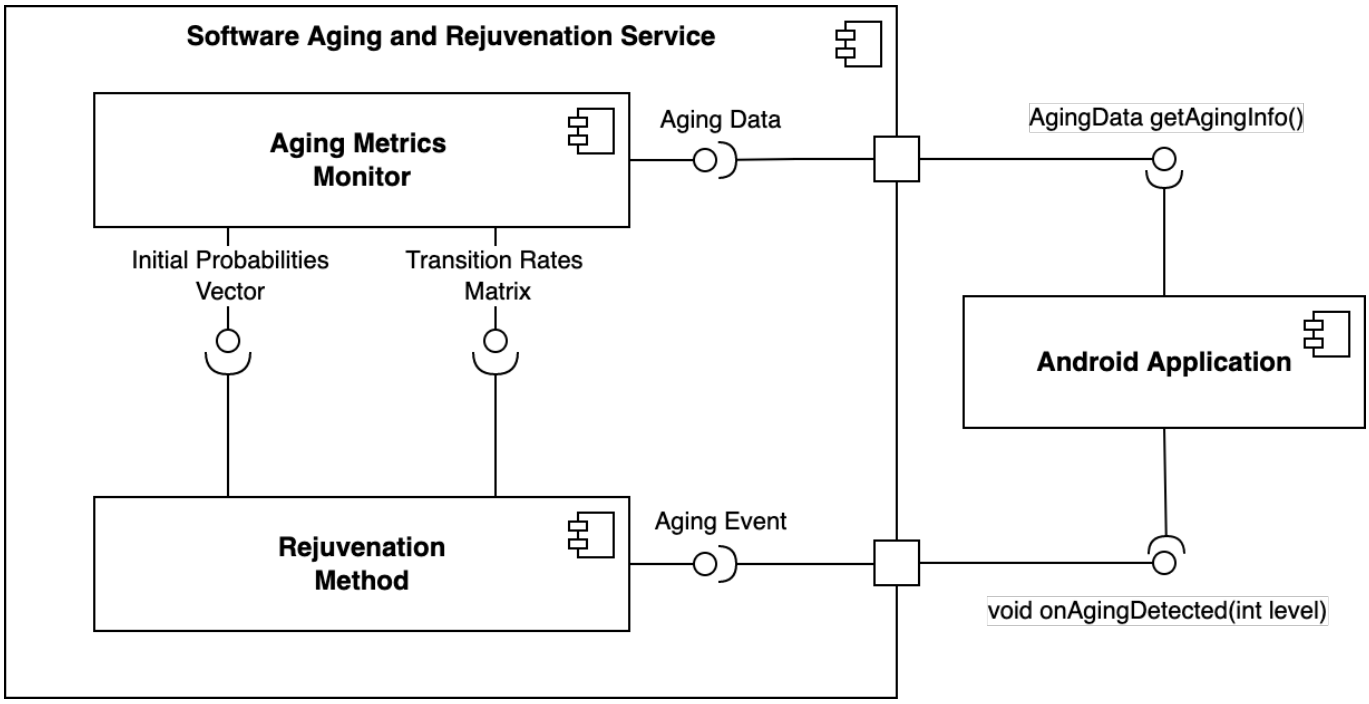


Рис. 4.7. UML-діаграма компонент для реалізації методу омолодження ПЗ для ОС Android

Основними завданнями модуля Aging Metrics Monitor є:

- постійний збір значень метрик, які визначені множинами M_{Aging}, M_{Using} та $M_{Battery}$ в розділі 3.3;

- визначення поточного стану старіння системи та забезпечення інтерфейсу доступу до початкового вектору імовірностей $P(0)$;

- визначення інтенсивностей переходів між станами та забезпечення інтерфейсу доступу до матриці інтенсивностей Q.

Регулярний моніторинг та акумулювання системних даних має враховувати як і достатність об'єму зібраних даних для формування статистики, так і низькі витрати апаратних ресурсів на виконання цієї процедури. Вимірювання коректних метрик, які з високою достовірністю відображають процеси старіння, має бути забезпечено на рівні сервісу ОС, щоб уникнути таких перетворень даних, які реалізовані і описані в розділі 4.2. Вибір оптимального інтервалу між вимірюваннями є важливим завданням, що потребує подальших досліджень, оскільки малий інтервал між записами вимагає виділення більшого об'єму пам'яті в сховищі даних, однак надто великий інтервал може негативно вплинути на точність моделювання під час прогнозування часу омолодження. Наприклад, наступний набір часових інтервалів може бути використано для дослідження ефективності роботи методу та оптимізації модуля моніторингу: $M_{Time} = \{1\ хв., 5\ хв., 10\ хв., 30\ хв., 60\ хв.\}$.

Модуль Rejuvenation Method є основним компонентом і може бути представлений у вигляді системного фонового процесу, який ініціалізується під час запуску мобільного пристрою. Основними завданнями модуля є:

- безпосереднє виконання методу омолодження, який описано в розділі 3.4 (рис. 3.17);

- прогнозування часу до відмови через старіння, наприклад, використовуючи методи регресійного моделювання чи аналітичні моделі на основі ланцюгів Маркова;

- прогнозування оптимального часу виконання омолодження з урахуванням обмежень (2.7), визначених на основі комплексної моделі представленої в розділі 2.4.

В межах даної роботи виконано програмну реалізацію запропонованих моделей, яку представлено в додатку Г. Кожна окрема модель представлена класом, успадкованим від базового класу `Model`, який дозволяє гнучко міняти конкретну реалізацію як для виконання експериментальних симуляцій процесу старіння та

омолодження, так і для використання в майбутніх програмних реалізаціях методу омолодження:

```
abstract class Model() {

    abstract fun modelName(): String
    abstract fun namesOfStates(): List<String>
    abstract fun initialProbabilities(): List<Double>
    abstract fun method(): Method

fun solve(initialTime: Double, intervalTime: Double, maxTime: Double): 
List<List<Double>> {
        this.initialTime = initialTime
        this.initialTime = intervalTime
        this.maxTime = maxTime
        val initialProbabilities = initialProbabilities()
        result = method().solve(initialTime, intervalTime, maxTime, 
initialProbabilities)
        return result
    }
}
```

Абстрактні функції `modelName()` та `namesOfStates()` дозволяють конкретизувати опис моделей з допомогою відповідних метаданих про назву моделі та її стани системи, що може бути використано для візуального чи табличного представлення.

Абстрактна функція `initialProbabilities()` дозволяє отримати вектор початкових імовірностей перебування в можливих станах моделі $P(0)$.

Абстрактна функція `method(): Method` дозволяє параметризувати конкретний тип реалізації основного методу моделювання. Наприклад, конкретною реалізацією абстрактного класу `Method` для визначення розв'язку систем диференційних рівнянь в даній роботі є клас `RungeKuttaMethod`.

Основна функція `solve()` класу `Model` дозволяє виконувати моделювання процесу старіння та омолодження для інтервалу часу заданого з допомогою аргументів `initialTime` та `maxTime`, а також, кроком обчислень `intervalTime`. Отриманий результат виконання функції – це двовимірна матриця, яка описує вектори імовірностей перебування системи в різних станах та у різний час.

Запропоновані моделі старіння та омолодження та метод омолодження ПЗ для вирішення проблем впливу старіння на користувацький досвід передбачають визначення активності користувача під час планування та виконання процедури омолодження. Під час активного використання пристрою користувачем метод омолодження передбачає відображення повідомлення користувачу із запитом виконати «холодне» омолодження ПЗ шляхом перезавантаження мобільного пристрою і користувач має можливість відмовитись від його виконання в користь продовження роботи із пристроєм в стані старіння, приймаючи можливі ризики відмови. В інших випадках методом передбачається виконання «теплого» омолодження ПЗ, що може допомогти відтермінувати відмову старіння. В обох випадках мова йде про, так зване, активне омолодження ПЗ на рівні ОС. Однак, крім такого підходу можна застосовувати активне омолодження або превентивні заходи на рівні користувацьких застосунків.

Важливими характеристиками будь-яких користувацьких застосунків та сервісів є надійність та стійкість до зовнішніх умов, таких як продуктивність ОС та доступність оперативної пам’яті, зокрема, до умов старіння ПЗ. Задачі, які вимагають складних обчислень чи значної кількості RAM, можуть виконуватись неефективно чи взагалі їх виконання може спричинити відмову в умовах старіння. Вирішенням таких проблем може бути застосування механізмів, які б дозволили алгоритмам застосунків приймати рішення про доцільність виконання тих чи інших дій, чи оптимізувати виконання задач під реальні умови продуктивності та старіння системи. Наприклад, складні застосунки для роботи із графікою чи ігрові застосунки у відповідь на повідомлення про старіння системи могли б адаптувати роботу своїх алгоритмів для мінімізації навантаження на систему зі своєї сторони і, тим самим, сповільнити процес старіння та покращити користувацький досвід. Ще одним прикладом може бути відображення повідомлень про старіння користувачу безпосередньо в застосунку, яке б інформувало, що система піддається впливу старіння і для покращення продуктивності потрібно, щоб користувач виконав певні дії, наприклад, власноруч закрив застосунки чи перезавантажив мобільний пристрій.

Для вирішення задачі активного омолодження на рівні користувацьких застосунків та превентивної протидії процесам старіння в запропонованих компонентах вимірювання метрик старіння та виконання омолодження передбачається наявність публічного інтерфейсу, доступного для розробників користувацького ПЗ. Цей інтерфейс можна представити двома основними методами:

- `AgingData getAgingInfo()` – функція доступу до підмодуля вимірювання метрик старіння, яка повертає об’єкт `AgingData` із відомостями про поточний стан старіння системи, час до відмови старіння та іншу можливу контекстну інформацію;

- `void onAgingDetected(AgingData)` – функція зворотного виклику, яка викликається ОС в той момент, коли модуль омолодження ПЗ виконує планування омолодження та при умові, що у системі спостерігаються процеси старіння, що можуть призводити до відмови.

Однією із основних причин старіння ПЗ є неправильне управління пам’яттю як на рівні ОС, так і на рівні користувацьких застосунків, що призводить до таких помилок, як витоки пам’яті та її переповнення. Тому, для пояснення та обґрунтування доцільності використання запропонованих методів `getAgingInfo()` та `onAgingDetected(AgingData)` можна розглянути для прикладу відомі аналогічні засоби ОС Android, але для визначення стану використання оперативної пам'яті системою та виконуваним процесом. Середовище розробки ОС Android забезпечує розробника засобами [118], що дозволяють враховувати критичні ситуації, що пов’язані із недостатньою кількістю вільної пам’яті під час виконання користувацького застосунку.

Використання функції зворотного виклику `onTrimMemory(level: Int)`, що викликається операційною системою, дозволяє розробнику виконати очищення даних застосунку вручну в залежності від значення аргументу `level`, наприклад, при критично низькому обсязі вільної пам’яті (`TRIM_MEMORY_RUNNING_MODERATE`, `TRIM_MEMORY_RUNNING_LOW`, `TRIM_MEMORY_RUNNING_CRITICAL`), або в тому випадку, коли застосування переходить у фоновий режим і графічний інтерфейс

користувача більше не відображається на дисплеї мобільного пристрою (TRIM_MEMORY_UI_HIDDEN). У випадку старіння ПЗ використання рівнів старіння «Young», «Aging» та «Old», описаних в попередніх розділах, у якості значень, що передаються з допомогою функції зворотного виклику onAgingDetected, дозволяє програмісту приймати рішення про зміни налаштування параметрів алгоритмів застосування чи очищення внутрішніх даних.

В контексті використання пам'яті виклик методу getMemoryInfo() та перевірка булевого значення lowMemory дозволяє приймати рішення про доцільність виконання ресурсоємних обчислень, для яких об'єм доступної пам'яті може бути не достатній:

```
public void doSomethingMemoryIntensive() {
    ActivityManager.MemoryInfo memoryInfo = getAvailableMemory();
    if (!memoryInfo.lowMemory) {
        // Do memory intensive work ...
    }
}
```

В той самий час, отримання даних про стан старіння системи та прогнозований час до відмови через старіння можна забезпечити методом getAgingInfo(). Наприклад, реалізація подібної функції дозволила б приймати рішення про доцільність виконання довготермінової задачі в залежності від дозволеного часового порогу та поточного стану старіння:

```
public void performSomethingLongTerm(int timeTreshhold) {
    AgingData agingData = getAgingInfo();
    if (agingData.level == AgingLevel.YOUNG || timeTreshhold <
agingData.timeToAgingFailure) {
        // Do intensive and long-term work …
    }
}
```

4.4. Висновки до розділу 4

У даному розділі описано відому методологію дослідження старіння ПЗ в контексті ОС Android, яка використана в даній роботі для дослідження метрик старіння, факторів старіння та самих процесів старіння в ОС Android. Методологія

складається із шести етапів: визначення стратегії моніторингу, визначення алгоритму генерування робочого навантаження, планування стресових тестів, виконання стресових тестів, аналіз зібраних даних та представлення результатів експерименту.

Описано програмну реалізацію фреймворку для виконання стресового тестування мобільних застосунків ОС Android та виконання аналізу зібраних даних у відповідності до попередньо визначеної методології дослідження. Фреймворк складається із трьох взаємозалежних модулів: модуль виконання стресових тестів, модуль генерування наборів часових рядів метрик, модуль аналізу часових рядів.

Модуль виконання стресових тестів реалізує алгоритми генерування робочого навантаження та збір метрик старіння та інших системних даних. Виконання стресових тестів відбувається на мобільних пристроях на ОС Android, які з'єднані з допомогою USB-кабеля із ПК, який виконує управління тестом в середовищі Windows Power Shell.

Модуль генерування наборів часових рядів метрик виконує перетворення «сирих» даних отриманих із модуля стресового тестування в таблиці даних відповідних метрик. Модуль написано на мові Kotlin і виконується в середовищі віртуальної машини Java.

Модуль аналізу часових рядів застосовує статистичні методи, методи регресійного аналізу та методи аналізу часових рядів для табличних даних, отриманих із модуля генерування часових рядів метрик. Модуль написано на мові Python та виконується у середовищі Google Colab.

Розроблено структуру модулів програмного засобу омолодження ПЗ для ОС Android, в контексті якої можливо реалізувати запропонований в третьому розділі метод омолодження ПЗ. Визначено основні вимоги до розроблюваного комплексу у вигляді системного сервісу, що працюватиме у фоновому режимі. Крім того, запропоновано засоби превентивної протидії старінню та омолодження на рівні користувацьких застосувань шляхом використання таких методів системного сервісу старіння та омолодження як функція `AgingData getAgingInfo()` та функція зворотного виклику `onAgingDetected(int agingLevel)`.

ВИСНОВКИ

У дисертації розв’язано актуальну наукову задачу підвищення рівня надійності ПЗ мобільних систем шляхом визначення факторів, що впливають на процес його старіння, побудови математичних моделей старіння і омолодження ПЗ та розроблення відповідних засобів.

Основні наукові та практичні результати дисертаційної роботи:

1) Проведено огляд та аналіз літературних джерел відповідно до теми дисертаційного дослідження. Виконано аналіз і опис основних тверджень поняття старіння та омолодження ПЗ. Проведено аналіз і порівняння найпоширеніших моделей старіння та омолодження ПЗ, а також, методів протидії негативним ефектам явища старіння.

2) Запропоновано аналітичні моделі старіння та омолодження, які враховують як активність використання мобільного пристрою користувачем, так і механізми «теплого» та «холодного» омолодження, різні рівні старіння та можливі стратегії виконання процедури омолодження, а також, фактор рівня заряду батареї. Для математичного опису процесу використано апарат ланцюгів Маркова з неперервним часом розподілу. Моделі старіння та омолодження можуть бути застосовані для оцінки показників якості існуючого ПЗ, дозволяють отримувати вирази для показників ефективності омолодження ПЗ, а також, можуть використовуватись для проектування та вибору параметрів методу омолодження ПЗ, що дає змогу формувати технічне завдання на його розроблення. Для запропонованої комплексної моделі старіння та омолодження визначено умови пошуку оптимального часу виконання омолодження з урахуванням наборів працездатних та непрацездатних станів системи, де непрацездатні стани включають як стани відмови, так і стани простою під час виконання омолодження.

3) Моделювання процесу старіння та омолодження з урахуванням різних рівнів старіння ПЗ та стратегій омолодження показали, що ефективною стратегією є повторюване виконання процедури омолодження в стані «старіння», яке дозволяє збільшити ймовірність перебування в стані «активного використання пристрою в

молодому стані» в 1,37-1,77 рази порівняно із симуляцією без виконання омолодження.

4) Моделювання процесу старіння та омолодження з урахуванням «теплого» та «холодного» механізмів омолодження показало, що у випадку «теплого» омолодження спостерігається більша імовірність перебування в стані «активного використання пристрою в молодому стані», ніж у випадку «холодного» омолодження. Значення різниці може залежати від реальних умов старіння та омолодження, тому врахування механізму омолодження є важливим завданням.

5) Моделювання процесу старіння та омолодження з урахуванням фактору заряду батареї показало, що рівень заряду може мати вагомий вплив на результат прогнозу часу виконання омолодження ПЗ. Зокрема, виконані симуляції підтверджують, що модель з урахуванням заряду батареї дозволяє отримати прогнозований час виконання омолодження в 4,5 рази менший, ніж аналогічний прогноз без урахування того факту, що батарея розрядиться швидше, ніж відбудеться омолодження.

6) Запропоновано та експериментально перевірено дві нові метрики GUI в контексті старіння ПЗ: FDT та JFC. Метрика FDT може бути використана у більшій кількості реальних сценаріїв використання ПЗ, ніж метрика ALT, і дозволяє отримувати неперервні часові ряди даних. Наприклад, у виконаних експериментах метрика FDT дозволяє отримати в 4 рази більше записів, ніж метрика ALT. Крім того, метрика FDT дозволяє визначати інтенсивність використання пристрою користувачем в різні моменти часу.

7) Досліджено нові фактори в контексті старіння, а саме, крос-платформові застосунки на базі фреймворку Flutter та сценарій використання мобільного пристрою із паузами під час генерування робочого навантаження. Показано, що крос-платформові застосунки Flutter можуть бути більш вразливими до ефектів старіння в контексті продуктивності UI, ніж стандартні native застосунки. Як для Flutter застосунків, так і для native застосунків тренд збільшення метрики FDT спостерігається у 2 із 6 виконаних експериментів, однак, абсолютні значення метрики відображення кадрів UI для Flutter значно перевищують native – приблизно вдвічі.

Сценарій використання із паузами характерний тим, що процес старіння може продовжуватись навіть без взаємодії користувача із UI. Зокрема, в системному процесі system_server продовжує спостерігатись позитивний тренд збільшення метрики PSS, що говорить про необоротність процесів старіння в ОС та необхідність застосування омолодження саме в момент пауз активності для покращення досвіду користувача та попередження виникнення відмов старіння.

8) На основі виконаних теоретичних та практичних досліджень запропоновано і описано метод омолодження ПЗ для ОС Android, який використовує комплексну модель старіння та омолодження для прогнозування часу виконання процедури омолодження. Метод, так само як і модель, визначає поступові рівні старіння системи, дозволяє будувати різні стратегії омолодження, враховує різні механізми процедури омолодження та враховує такі фактори як активність використання мобільного пристрою користувачем та рівень заряду батареї. Метод дозволяє зменшити ймовірність виникнення відмов старіння, а також, покращити досвід користувача та характеристики продуктивності ПЗ.

9) Розроблено фреймворк для виконання стресового тестування мобільних застосунків ОС Android та виконання аналізу зібраних даних у відповідності до описаної в даній роботі відомої методології дослідження явища старіння. Фреймворк складається із трьох взаємозалежних модулів: модуль виконання стресових тестів, модуль генерування наборів часових рядів метрик, модуль аналізу часових рядів. Розроблений фреймворк може використовуватись на етапі тестування ПЗ.

10) Розроблено структуру модулів програмного засобу омолодження ПЗ для ОС Android, в межах якої можливо реалізувати запропонований метод омолодження ПЗ, визначено основні вимоги до розроблюваного комплексу у вигляді системного сервісу, що працюватиме у фоновому режимі на етапі експлуатації ПЗ. Публічний інтерфейс для взаємодії із описаними модулями старіння та омолодження може бути використаний на етапі проектування та розроблення користувацьких застосувань, наприклад: функція для отримання даних про стан старіння системи та функція зворотного виклику на подію про виявлення старіння в системі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1]. Яковина В. С., Угриновський Б. В. Старіння програмного забезпечення в контексті його надійності: огляд проблематики. Науковий вісник НЛТУ України. 2019. 29(5), 123-128. DOI: https://doi.org/10.15421/40290525

[2]. Яковина В. С., Угриновський Б. В. Старіння програмного забезпечення мобільних додатків: огляд проблематики. Науковий вісник НЛТУ України. 2020. 30(2), 107-112. DOI: https://doi.org/10.36930/40300219

[3]. Яковина В. С., Угриновський Б. В. Метрики інтерфейсу користувача для виявлення явища старіння програмного забезпечення в мобільній системі Android. Вісник НУЛП «Інформаційні системи та мережі». 2021. 9, 32-43. DOI: https://doi.org/10.23939/sisn2021.09.032

[4]. Яковина В. С., Угриновський Б. В. Дослідження системних процесів та користувацьких додатків операційної системи Android в контексті старіння програмного забезпечення. Вісник ХНУ: Технічні науки. 2021. 295(2), 64-70. DOI: https://www.doi.org/10.31891/2307-5732-2021-295-2-64-70

[5]. Yakovyna V. S., Uhrynovskyi B. V. Extended software aging and rejuvenation model for Android operating system considering different aging levels and rejuvenation procedure types. Computer systems and information technologies. 2021. 3, 116-124. DOI: https://doi.org/10.31891/CSIT-2021-5-9

[6]. Яковина В. С., Угриновський Б. В. Метод омолодження програмного забезпечення для операційної системи Android з використанням комплексної моделі його старіння на підставі ланцюга Маркова. Науковий вісник НЛТУ України. 2021. 31(6), 97-103. DOI: https://doi.org/10.36930/40310615

[7]. Yakovyna V. S., Uhrynovskyi B. V. Android software aging and rejuvenation model considering the battery charge. Radio Electronics, Computer Science, Control. 2021. (4), 140-148. DOI: https://doi.org/10.15588/1607-3274-2021-4-13

[8]. Yakovyna V. S., Uhrynovskyi B. V., Bachkay O. Software Failures Forecasting by Holt - Winters, ARIMA and NNAR Methods. *IEEE 14th International Conference on Computer Sciences and Information Technologies*. 2019. DOI: https://doi.org/10.1109/STC-CSIT.2019.8929863

[9]. Yakovyna V. S., Uhrynovskyi B. V. User-Perceived Response Metrics in Android OS for Software Aging detection. *IEEE 15th International Conference on Computer Sciences and Information Technologies*. 2020. DOI: https://doi.org/10.1109/CSIT49958.2020.9322031

[10]. Яковина В. С., Угриновський Б. В. Модель старіння та омолодження програмного забезпечення для платформи Android. *XX Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2020. 5-8 червня.

[11]. Яковина В. С., Угриновський Б. В. Засоби протидії явищу старіння програмного забезпечення в операційній системі Android. *XXI Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2021. 3-7 червня.

[12]. Yakovyna V. S., Uhrynovskyi B. V. Aging of Native and Flutter Applications in Android OS in Various Usage Scenarios. *IEEE 14th International Conference on Computer Sciences and Information Technologies.* 2021. DOI: https://doi.org/10.1109/CSIT52700.2021.9648777

[13]. Huang Y. Software rejuvenation: Analysis, module and applications / Y. Huang, C. Kintala, N. Kolettis, N. D. Fulton // 25th Symposium on Fault Tolerant Computing, 27-30 June 1995: proceedings. – Pasadena, California, 1995. – P.381–390. https://doi.org/10.1109/FTCS.1995.466961

[14]. Parnas D. L. Software aging / D. L. Parnas // 16th International Conference on Software Engineering, 1994: proceedings. – P.279-287. https://doi.org/10.1109/ICSE.1994.296790

[15]. Bernstein L. and Kintala C. M. R. (2004). Software Rejuvenation. CrossTalk, 6(8), 23-26.

[16]. A. Avritzer and E. J. Weyuker, Monitoring Smoothly Degrading Systems for Increased Dependability. Empirical Software Eng. J., 2 (1): 59–77, 1997.

[17]. Marshall E. Fatal error: How patriot overlooked a scud. *Science*. 1992. Vol. 255, No. 5050. P. 1347.

[18]. Bernstein L. Innovative technologies for preventing network outages. *AT&T Technical Journal.* 1993. Vol. 72, No. 4. P. 4-10. https://doi.org/10.1002/j.1538-7305.1993.tb00545.x

[19]. Y. Bao, X. Sun, and K. S. Trivedi. "A workload-based analysis of software aging and rejuvenation," IEEE Transactions on Reliability, vol. 55, no. 3, pp. 541–548, 2005. https://doi.org/10.1109/TR.2005.853442

[20]. M. Shereshevsky, J. Crowell, B. Cukic, V. Gandikota, and Y. Liu. "Software aging and multifractality of memory resources," In Proc. IEEE International Conference on Dependable Systems and Networks, pp. 721–730, 2003.

[21]. M. Grottke, L. Li, K. Vaidyanathan, and K. S. Trivedi. "Analysis of software aging in a web server," IEEE Transactions on Reliability, vol. 55, no. 3, pp. 411–420, 2006. https://doi.org/10.1109/TR.2006.879609

[22]. R. Matias and P. J. Freitas. "An experimental study on software aging and rejuvenation in web servers," In Proc. 30th Annual International Computer Software and Applications Conference, vol. 1, pp. 189–196, 2006. https://doi.org/10.1109/COMPSAC.2006.25

[23]. L. Silva, H. Madeira, and J. G. Silva. "Software aging and rejuvenation in a SOAP-based server," In Proc. Fifth IEEE International Symposium on Network Computing and Applications, pp. 56–65, 2006.

[24]. M. Grottke, R. M. Jr, and K. S. Trivedi, "The fundamentals of software aging," in Proc. 1st Int. Workshop on Software Aging and Rejuvenation IEEE 19th International Symposium on Software Reliability Engineering, 2008, pp. 1–6. https://doi.org/10.1109/ISSREW.2008.5355512

[25]. S. Russo and N. Federico, "The dual nature of software aging twenty years of software aging research," in 2014 IEEE Int. Symp. on Software Reliability Engineering Workshops, 2014, pp.431-432.

[26]. ISO 25010. [Електронний ресурс] // Режим доступу: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3&limitstart=0/

[27]. S. Ahamad, “Study of software aging issues and prevention solutions,” Int. J. Comput. Sci. Inf. Secur., vol. 14, no. 8, pp. 307–313, 2016.

[28]. Polovko A.M., Gurov S.V. (2008). Osnovy teorii nadezhnosti [Fundamentals of Reliability Theory]. St.Petersburg: BHV-Petersburg, 704. [in Russian]

[29]. P. Tobias and D. Trindade. Applied Reliability, 2nd edition. Kluwer Academic Publishers, Boston, 1995.

[30]. A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. “Basic concepts and taxonomy of dependable and secure computing,” IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11–33, 2004. https://doi.org/10.1109/TDSC.2004.2

[31]. J. D. Musa. “Operational profiles in software reliability en- gineering,” IEEE Software, vol. 10, no. 2, pp. 14–32, 1993. https://doi.org/10.1109/52.199724

[32]. M. Grottke and K. S. Trivedi. “Software faults, software aging, and software rejuvenation,” Journal of the Reliability Association of Japan, vol. 27, no. 7, pp. 425–438, 2005.

[33]. M. Grottke and K. S. Trivedi. “Fighting bugs: Remove, retry, replicate and rejuvenate,” IEEE Computer, vol. 40, no. 2, pp. 107–109, 2007.

[34]. Abdullah Z. H., Yahaya J. H., Mansor Z., & Deraman A. (2017). Software Ageing Prevention from Software Maintenance Perspective – A Review. Journal of Telecommunication, Electronic and Computer Engineering, 9(3-4), 93-96.

[35]. Wu H. Software aging in mobile devices: Partial computation offloading as a solution / H. Wu, K. Wolter // IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). – 2015. – P. 125–131. https://doi.org/10.1109/ISSREW.2015.7392057

[36]. J. H. Yahaya, A. Deraman, and Z. H. Abdullah, “Evergreen software preservation : The anti-ageing model,” in ICC ’16 Proc. Int. Conf. Internet things Cloud Comput. United Kingdom, 2016, pp. 1–6.

[37]. Z. H. Abdullah, and J. Yahaya, “Anti-aging factors for application software - a preliminary study,” in Int. Symp. on Research in Innovation and Sustainability 2014 (ISoRIS ’14), 2014, pp. 15–16.

[38]. J. H. Yahaya, and A. Deraman, “Towards the anti-ageing model for application software,” in 2012 International Conference on Electrical Engineering and Informatics (ICEEI), pp. 388-393, 2012

[39]. J. H. Yahaya, A. Deraman, and Z. H. Abdullah, “Evergreen software perservation : the conceptual framework of anti-ageing model,” in Information Science and Applications, K. J. Kim, Ed. 2015, pp. 899- 906

[40]. J. H. Yahaya, Z. N. Z. Abidin, and A. Deraman, “Perspective and perception on software ageing: The empirical study,” in 10th Int. Conf. Comput. Sci. Educ. ICCSE 2015, 2015, pp. 365–370.

[41]. V. Basili and B. Perricone, “Software Errors and Complexity: An Empirical Investigation,” Communications of the ACM, vol. 27, no. 1, 1984. https://doi.org/10.1145/69605.2085

[42]. Cotroneo D., Natella R., & Pietrantuono R. (2011). Is Software Aging related to Software Metrics. 2010 IEEE Second International Workshop on Software Aging and Rejuvenation. https://doi.org/10.1109/WOSAR.2010.5722096

[43]. Cotroneo D., Natella R., Pietrantuono R., & Russo S. (2014). A Survey of Software Aging and Rejuvenation Studies. ACM Journal on Emerging Technologies in Computing Systems, 10(1), article 8. https://doi.org/10.1145/2539117

[44]. Valentim N. A., Macedo A., & Matias R. (2016). A Systematic Mapping Review of the First 20 Years of Software Aging and Rejuvenation Research. 2016 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). DOI: https://doi.org/10.1109/ISSREW.2016.42

[45]. Garg S., Kintala C., Huang Y., and Trivedi K. Minimizing completion time of a program by checkpointing and rejuvenation. Performance Evaluation Review. 1996. Vol. 24, No. 1. P. 252–261.

[46]. Garg S., Puliafito A., Telek M., and Trivedi K. Analysis of preventive maintenance in transactions based software systems. IEEE Transactions on Computers. 1998. Vol. 47, No. 1. P. 96-107. DOI: https://doi.org/10.1109/12.656092

[47]. Xie W., Hong Y., & Trivedi K. (2004). Software rejuvenation policies for cluster systems under varying workload. Proceedings of 10th IEEE Pacific Rim International

Symposium on Dependable Computing. DOI: https://doi.org/10.1109/PRDC.2004.1276563

[48]. Myint M. and Thein T. Availability improvement in virtualized multiple servers with software rejuvenation and virtualization. 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement. 2010. P. 156–162. DOI: https://doi.org/10.1109/SSIRI.2010.19

[49]. Kourai K. and Chiba S. A fast rejuvenation technique for server consolidation with virtual machines. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07). 2007. DOI: https://doi.org/10.1109/DSN.2007.6

[50]. Du X., Qi Y., Hou D., Chen Y., and Zhong X. Modeling and performance analysis of software rejuvenation policies for multiple degradation systems. 2009 33rd Annual IEEE International Computer Software and Applications Conference. 2009. P. 240–245. DOI: https://doi.org/10.1109/COMPSAC.2009.39

[51]. Pfening A., Garg S., Puliafito A., Telek M., & Trivedi K. (1996). Optimal software rejuvenation for tolerating soft failures. Performance Evaluation, 27/28, 491–506. https://doi.org/10.1016/S0166-5316(96)90042-5

[52]. Koutras V. and Platis A. Applying partial and full rejuvenation in different degradation levels. 2011 IEEE Third International Workshop on Software Aging and Rejuvenation. 2011. P. 20 –25. DOI: https://doi.org/10.1109/WoSAR.2011.14

[53]. Okamura H. & Dohi T. (2011). A pomdp formulation of multistep failure model with software rejuvenation. Proceedings of IEEE Third International Workshop on Software Aging and Rejuvenation, 14–19. https://doi.org/10.1109/WoSAR.2011.11

[54]. Garg S., Puliafito A., Telek M., and Trivedi K. Analysis of software rejuvenation using markov regenerative stochastic petri net. Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95. 1995. DOI: https://doi.org/10.1109/ISSRE.1995.497656

[55]. Wang D., Xie W., & Trivedi K. (2007). Performability analysis of clustered systems with rejuvenation under varying workload. Performance Evaluation, 64(3), 247–265. https://doi.org/10.1016/j.peva.2006.04.002

[56]. Vaidyanathan K., Harper R., Hunter S., & Trivedi K. (2001). Analysis and implementation of software rejuvenation in cluster systems. Performance Evaluation Review, 29(1), 62–71. https://doi.org/10.1145/378420.378434

[57]. Salfner F. and Wolter K. Analysis of service availability for time-triggered rejuvenation policies. Journal of Systems and Software. 2010. Vol. 83, No. 9. P. 1579–1590. DOI: https://doi.org/10.1016/j.jss.2010.05.022

[58]. Andrade E., Machida F., Kim D., and Trivedi K. Modeling and analyzing server system with rejuvenation through sysml and stochastic reward nets. Proceedings of the 6th International Conference on Availability, Reliability and Security, ARES. 2011. P. 161–168. DOI: https://doi.org/10.1109/ARES.2011.28

[59]. Delligatti, Lenny. SysML Distilled: A Brief Guide to the Systems Modeling Language. — Addison-Wesley Professional, 2013. — ISBN 978-0-321-92786-6.

[60]. Huang Y. and Kintala C. Software implemented fault tolerance: Technologies and experience. The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993.

[61]. Garg S., Van Moorsel A., Vaidyanathan K., and Trivedi K. A methodology for detection and estimation of software aging. Proceedings Ninth International Symposium on Software Reliability Engineering. 1998. DOI: https://doi.org/10.1109/ISSRE.1998.730892

[62]. Li L., Vaidyanathan K., & Trivedi K. (2002). An approach for estimation of software aging in a web server. Proceedings International Symposium on Empirical Software Engineering. https://doi.org/10.1109/ISESE.2002.1166929

[63]. Cotroneo D., Natella R., Pietrantuono R., & Russo S. (2010). Software aging analysis of the linux operating system. 2010 IEEE 21st International Symposium on Software Reliability Engineering. https://doi.org/10.1109/ISSRE.2010.24

[64]. Cotroneo D., Orlando S., Pietrantuono R., and Russo S. A measurement-based ageing analysis of the jvm. Software Testing Verification and Reliability. 2011. Vol. 23, No. 3. P. 199-239. DOI: https://doi.org/10.1002/stvr.467

[65]. Magalhaes J. & Silva L. (2010). Prediction of performance anomalies in web-applications based-on software aging scenarios. 2010 IEEE Second International

Workshop on Software Aging and Rejuvenation. https://doi.org/10.1109/WOSAR.2010.5722095

[66]. Araujo J., Matos R., Maciel P., Vieira F., Matias R., and Trivedi K. Software rejuvenation in eucalyptus cloud computing infrastructure: A method based on time series forecasting and multiple thresholds. IEEE Third International Workshop on Software Aging and Rejuvenation. 2011. P. 38-43. DOI: https://doi.org/10.1109/WoSAR.2011.18

[67]. Eucalyptus. [Електронний ресурс] // Режим доступу: www.eucalyptus.cloud

[68]. Hoffman G., Trivedi K., and Malek M. A best practice guide to resource forecasting for computing systems. IEEE Transactions on Reliability. 2007. Vol. 56, No. 4. P. 615-628.

[69]. Cassidy K., Gross K., and Malekpour A. Advanced pattern recognition for detection of complex software aging phenomena in online transaction processing servers. Proceedings International Conference on Dependable Systems and Networks. 2002. DOI: https://doi.org/10.1109/DSN.2002.1028933

[70]. Alonso J., Belanche L., & Avresky D. (2011). Predicting software anomalies using machine learning techniques. Proceedings of 2011 IEEE International Symposium on Network Computing and Applications, 163–170. https://doi.org/10.1109/NCA.2011.29

[71]. Silva L., Alonso J., & Torres J. (2009). Using virtualization to improve software rejuvenation. IEEE Transactions on Computers, 58(11), 1525-1538. https://doi.org/10.1109/TC.2009.119

[72]. Vaidyanathan K. and Trivedi K. A measurement-based model for estimation of resource ex- haustion in operational software systems. Proceedings 10th International Symposium on Software Reliability Engineering. 1999. DOI: https://doi.org/10.1109/ISSRE.1999.809313

[73]. Vaidyanathan K. and Trivedi K. 2005. A comprehensive model for software rejuvenation. IEEE Transactions on Dependable and Secure Computing. 2005. Vol. 2, No. 2. P.124-137. DOI: https://doi.org/10.1109/TDSC.2005.15

[74]. Andrzejak A. and Silva L. Deterministic models of software aging and optimal rejuvenation schedules. 10th IFIP/IEEE International Symposium on Integrated Network Management. 2007. P. 159–168. DOI: https://doi.org/10.1109/INM.2007.374780

[75]. Bovenzi A., Cotroneo D., Pietrantuono R., and Russo S. Workload characterization for software aging analysis. IEEE 22nd International Symposium on Software Reliability Engineering. P. 240-249. DOI: https://doi.org/10.1109/ISSRE.2011.18

[76]. Carrozza G., Cotroneo D., Natella R., Pecchia A., and Russo S. Memory leak analysis of mission-critical middleware. Journal of Systems and Software. 2010. Vol. 83, No. 9, P. 1556-1567. DOI: https://doi.org/10.1016/j.jss.2010.05.027

[77]. Xu G., Bond M., Qin F., and Rountev A. Leakchaser: Helping programmers narrow down causes of memory leaks. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2011. Vol. 46, No. 6. P. 270-282. DOI: https://doi.org/10.1145/1993316.1993530

[78]. Xu G. and Rountev A. Precise memory leak detection for java software using container profiling. Proceedings - International Conference on Software Engineering. 2008. Vol. 22, No. 3. P. 151–160. DOI: https://doi.org/10.1145/2491509.2491511

[79]. Weimer W. Exception-handling bugs in java and a language extension to avoid them. Lecture Notes in Computer Science 4119 LNCS. 2006. P. 22–41.

[80]. Heine D. and Lam M. Static detection of leaks in polymorphic containers. Proceedings - International Conference on Software Engineering. 2006. P. 252-261. DOI: https://doi.org/10.1145/1134285.1134321

[81]. Grottke M., Nikora A., and Trivedi K. An empirical investigation of fault types in space mis- sion system software. 2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN). 2010. DOI: https://doi.org/10.1109/DSN.2010.5544284

[82]. Dohi, T., & Trivedi, K., & Avritzer, A. (2020). Handbook of software aging and rejuvenation. World Scientific Publishing Co Pte Ltd, 424. https://doi.org/10.1142/11673

[83]. Adams E. Optimizing preventive service of software products. IBM Journal of Research and Development. 1984. Vol. 28, No. 1. P. 2-14.

[84]. Wand Y.-M., Huang Y., Vo K.-P., Chung P.-Y., and Kintala C. 1995. Checkpointing and its applications. Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers. 1995. DOI: https://doi.org/10.1109/FTCS.1995.466999

[85]. Trivedi, K. S. and Vaidyanathan, K. 2007. Software Aging and Rejuvenation. Wiley Encyclopedia of Computer Science and Engineering.

[86]. Bruneo, Dario; Distefano, Salvatore; Longo, Francesco; Puliafito, Antonio; Scarpa, Marco (2013). "Workload-Based Software Rejuvenation in Cloud Systems". IEEE Transactions on Computers. 62 (6): 1072–1085. doi:10.1109/TC.2013.30.

[87]. Trivedi, Kishor S.; Vaidyanathan, Kalyanaraman (2004-01-01). Reis, Ricardo (ed.). Software Rejuvenation - Modeling and Analysis. IFIP International Federation for Information Processing. Springer US. pp. 151–182. doi:10.1007/1-4020-8159-6_6. ISBN 978-1-4020-8158-3.

[88]. Guo C. Use Two-Level Rejuvenation to Combat Software Aging and Maximize Average Resource Performance / C. Guo, H. Wu, X. Hua, D. Lautnery, S. Ren // IEEE International Conference on High Performance Computing and Communications. – 2015. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.306

[89]. Cotroneo D. A Comprehensive Study on Software Aging across Android Versions and Vendors / D. Cotroneo, A. K. Iannillo, R. Natella, R. Pietrantuono // Empirical Software Engineering. – 2020. – Volume 25(3). P. 3357-3395.

[90]. A.K. Maji, K. Hao, S. Sultana, and S. Bagchi, “Characterizing failures in mobile OSes: a case study with Android and Symbian,” International Symposium on Software Reliability Engineering. IEEE Computer Society, 2010, pp. 249-258.

[91]. D. Cotroneo, F. Fucci, A.K. Iannillo, R. Natella, and R. Pietrantuono, “Software aging analysis of the android mobile os,” IEEE International Symposium on Software Reliability Engineering (ISSRE), 2016, pp. 478-489.

[92]. IDC: smartphone OS market share. [Електронний ресурс] // Режим доступу: http://www.idc.com/promo/smartphone-market-share/os

[93]. Y. Qiao, Z. Zheng, and F.Y. Qin, “An empirical study of software aging manifestation in android,” International Symposium on Software Reliability Engineering Workshops (ISSREW), IEEE, 2016, pp. 84-90.

[94]. Araujo J., Alves V., Oliveira D., Dias P., Silva B., Maciel P. (2013). An Investigative Approach to Software Aging in Android Applications. 2013 IEEE International Conference on Systems, Man, and Cybernetics. https://doi.org/10.1109/SMC.2013.213

[95]. UI/Application Exerciser. [Електронний ресурс] // Режим доступу: https://developer.android.com/studio/test/monkey

[96]. Android Debug Bridge. [Електронний ресурс] // Режим доступу: https://developer.android.com/studio/command-line/adb

[97]. T. Hayashi and S. Ohta, “Performance degradation detection of virtual machines via passive measurement and machine learning,” International Journal of Adaptive Resilient and Autonomic Systems, 2014, vol. 5, no. 2, pp. 40-56.

[98]. J. Alonso, J. Torres, and R. Gavaldà, “Predicting web server crashed: A case study in comparing prediction algorithms,” the fifth Int. Conference on Autonomic and Autonomous Systems (ICSA), IEEE, 2009, pp, 264- 269.

[99]. J. Alonso, J. Torres, J.L. Berral, and R. Gavalda, “Adaptive on-line software aging prediction based on machine learning,” in Proc. of the 40th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN 2010). IEEE, 2010, pp. 507–516.

[100]. A. Andrzejak and L. Silva, “Using machine learning for non-intrusive modeling and prediction of software aging,” in Proc. of 2008 IEEE Network Operations and Management Symp. (NOMS 2008). IEEE, 2008, pp. 25–32.

[101]. Huo S., Zhao D., Liu X., Xiang J., Zhong Y., & Yu H. (2018). Using machine learning for software aging detection in Android system. 2018 Tenth International Conference on Advanced Computational Intelligence (ICACI). https://doi.org/10.1109/ICACI.2018.8377553

[102]. Cotroneo D., Simone L. D., Natella R., Pietrantuono R., & Russo S. (2019). A Configurable Software Aging Detection and Rejuvenation Agent for Android. 11th

International Workshop on Software Aging and Rejuvenation (WoSAR). https://doi.org/10.1109/ISSREW.2019.00078

[103]. Xianga J., Wenga C., Zhaoa D., Tiana J., Xionga S., Lia L., Andrzejak A. (2019). A New Software Rejuvenation Model for Android. 2018 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). https://doi.org/10.1109/ISSREW.2018.00021

[104]. Stochastic Petri net. [Електронний ресурс] // Режим доступу: https://en.wikipedia.org/wiki/Stochastic_Petri_net

[105]. G. Bucci, L. Carnevali, L. Ridi, and E. Vicario, "Oris: a tool for modeling, verification and evaluation of real-time systems," International Journal on Software Tools for Technology Transfer, vol. 12, pp. 391-403, September 01 2010

[106]. Norris J. R. Markov Chains / J. R. Norris // Cambridge University Press, 1997.

[107]. Dumpsys – Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/studio/command-line/dumpsys

[108]. Memory allocation among processes – Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/topic/performance/memory-management

[109]. Android Runtime (ART) and Dalvik. [Електронний ресурс] // Режим доступу: https://source.android.com/devices/tech/dalvik

[110]. Test UI Performance – Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/training/testing/performance

[111]. Flutter – Build apps for any screen. [Електронний ресурс] // Режим доступу: https://flutter.dev/

[112]. Activity – Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/reference/android/app/Activity

[113]. App startup time – Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/topic/performance/vitals/launch-time

[114]. Power Shell Documentation – Microsoft Docs. [Електронний ресурс] // Режим доступу: https://docs.microsoft.com/en-us/powershell/

[115]. Android Developers. Logcat command-line tool. [Електронний ресурс] // Режим доступу: https://developer.android.com/studio/command-line/logcat

[116]. GitHub. Android SAR Dataset Generator. [Електронний ресурс] // Режим доступу: https://github.com/bohdanuhryn/android_sar_dataset_generator

[117]. Colaboratory. Welcome to Colaboratory. [Електронний ресурс] // Режим доступу: https://colab.research.google.com/

[118]. Manage your app’s memory - Android Developers. [Електронний ресурс] // Режим доступу: https://developer.android.com/topic/performance/memory

ДОДАТКИ

Додаток А Набір даних для виконання тестових симуляцій процесу старіння та омолодження

SIM	Модель активності використання мобільного пристрою		Модель старіння ПЗ				Модель омолодження ПЗ				
	T_{SmAm}, хв.	T_{AmSm}, хв.	T_{YA}, год.	T_{AO}, год.	T_{OF}, год.	T_{FY}, хв.	T_{AR}, год.	T_{OR}, год.	T_{RY}, хв.	T_{SmAmR} хв.	T_{SmAmR} хв.
SIM-0	75	10	48	96	24	-	-	-	-	-	-
SIM-1	75	10	48	96	24	1	-	-	-	T_{SmAm}	T_{AmSm}
SIM-2	75	10	4	8	2	5	-	-	-	T_{SmAm}	T_{AmSm}
SIM-3	30	50	48	96	24	1	-	-	-	T_{SmAm}	T_{AmSm}
SIM-4	30	50	4	8	2	5	-	-	-	T_{SmAm}	T_{AmSm}
SIM-5	75	10	48	96	24	1	48	12	1	T_{SmAm}	T_{AmSm}
SIM-6	75	10	48	96	24	1	48	-	1	T_{SmAm}	T_{AmSm}
SIM-7	75	10	48	96	24	1	-	12	1	T_{SmAm}	T_{AmSm}
SIM-8	75	10	48	96	24	1	144	36	1	T_{SmAm}	T_{AmSm}
SIM-9	75	10	4	8	2	5	4	1	5	T_{SmAm}	T_{AmSm}
SIM-10	75	10	4	8	2	5	4	-	5	T_{SmAm}	T_{AmSm}
SIM-11	75	10	4	8	2	5	-	1	5	T_{SmAm}	T_{AmSm}
SIM-12	75	10	4	8	2	5	12	3	5	T_{SmAm}	T_{AmSm}
SIM-13	30	50	48	96	24	1	48	12	1	T_{SmAm}	T_{AmSm}
SIM-14	30	50	48	96	24	1	48	-	1	T_{SmAm}	T_{AmSm}

SIM-15	30	50	48	96	24	1	-	12	1	T_{SmAm}	T_{AmSm}
SIM-16	30	50	48	96	24	1	144	36	1	T_{SmAm}	T_{AmSm}
SIM-17	30	50	4	8	2	5	4	1	5	T_{SmAm}	T_{AmSm}
SIM-18	30	50	4	8	2	5	4	-	5	T_{SmAm}	T_{AmSm}
SIM-19	30	50	4	8	2	5	-	1	5	T_{SmAm}	T_{AmSm}
SIM-20	30	50	4	8	2	5	12	3	5	T_{SmAm}	T_{AmSm}
SIM-21	75	10	48	96	24	1	48	-	1	-	-
SIM-22	75	10	4	8	2	5	4	-	5	-	-
SIM-23	30	50	48	96	24	1	48	-	1	-	-
SIM-24	30	50	4	8	2	5	4	-	5	-	-

Додаток Б Конфігурація виконаних досліджень старіння ПЗ в ОС Android

<table>
<tr><th colspan="4">EXP1. Конфігурація дослідження метрик продуктивності GUI</th></tr>
<tr><td colspan="2"></td><td>Тести з примусовими перезапусками додатків</td><td>Тести без примусових перезапусків додатків</td></tr>
<tr><td colspan="2">Кількість тестів</td><td>4</td><td>4</td></tr>
<tr><td colspan="2">Тривалість виконання тесту</td><td colspan="2">2,5-3 години</td></tr>
<tr><td colspan="2">Інтервал запису метрик</td><td colspan="2">20-30 секунд</td></tr>
<tr><td colspan="2">Користувацькі додатки</td><td colspan="2">com.google.android.youtube
com.android.contacts
com.android.chrome
com.android.camera
com.google.android.apps.photos</td></tr>
<tr><td rowspan="2">Метрики</td><td>UI</td><td colspan="2">ALT
FDT
JFC</td></tr>
<tr><td>Системні</td><td colspan="2">PSS
Total RAM, Used RAM, Free RAM</td></tr>
<tr><td colspan="2">Версія Android</td><td colspan="2">7.0</td></tr>
<tr><td colspan="2">Модель мобільного пристрою</td><td colspan="2">Xiaomi Redmi Note 4, 3 Gb RAM, 2018</td></tr>
</table>

<table>
<tr><th colspan="6">EXP2. Конфігурація експерименту дослідження сценаріїв використання мобільного пристрою та типів користувацьких додатків</th></tr>
<tr><td colspan="3">Тривалість виконання тесту</td><td colspan="3">4,25 години</td></tr>
<tr><td colspan="3">Інтервал запису метрик</td><td colspan="3">30 секунд</td></tr>
<tr><td colspan="3">Кількість тестів</td><td colspan="3">3</td></tr>
<tr><td colspan="3">Група тестів</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td rowspan="2">Типи користувацьких додатків</td><td>Default Android - Native</td><td>com.google.android.youtube
com.android.chrome
com.android.camera
com.google.android.apps.photos
com.google.android.apps.maps</td><td>+</td><td></td><td>+</td><td></td></tr>
<tr><td>Cross-platform - Flutter</td><td>com.reflectlyApp
com.hamilton.app
com.ebay.motorsapp
com.spotlightsix.zentimerlite2
com.mgmresorts.mgmresorts</td><td></td><td>+</td><td></td><td>+</td></tr>
<tr><td rowspan="2">Сценарій використання</td><td colspan="2">Безперервне використання: постійне генерування робочого навантаження</td><td colspan="2">+</td><td colspan="2"></td></tr>
<tr><td colspan="2">Використання із паузами: 40 хв. генерування робочого навантаження – 30 хв. паузи</td><td colspan="2"></td><td colspan="2">+</td></tr>
<tr><td rowspan="2">Метрики</td><td colspan="2">UI</td><td colspan="4">ALT
FDT
JFC</td></tr>
<tr><td colspan="2">Системні</td><td colspan="4">PSS
RAM
GC</td></tr>
<tr><td colspan="3">Версія ОС Android</td><td colspan="4">7.1.2</td></tr>
<tr><td colspan="3">Модель мобільного пристрою</td><td colspan="4">Xiaomi Redmi 4X, 2017</td></tr>
</table>

Додаток В Лістинг модуля виконання стресових тестів та збору системних даних мобільного пристрою

Назва файлу: monitor.ps1.

Короткий опис: функції виконання утиліт командного рядка для читання системних даних Android із мобільного пристрою.

Вихідний код:

```
function InitOutputDirs
{
    param ([String] $Output)
    # Root dir
    CreateFileIfNotExists -Path ".\output" -Name $Output -Type "directory"
    # LogCat dir
    CreateFileIfNotExists -Path ".\output\$Output" -Name "logcat" -Type "directory"
    # Dumpsys dir
    CreateFileIfNotExists -Path ".\output\$Output" -Name "dumpsys" -Type "directory"
    # ProcTasks dir
    CreateFileIfNotExists -Path ".\output\$Output" -Name "proctasks" -Type "directory"
}

function BugReportsMonitor
{
    Param([String] $Output)
    Write-Host "RunBugReport call"
    $path = ".\output\$Output\bugReports"
    adb bugreport $path
}

function DumpsysServiceMonitor
{
    Param(
        [String] $Output,
        [String] $Service,
        [String[]] $Packages,
        [String] $OptionalParams = ""
    )
    Write-Host "Dumpsys $Service call"
    $path = ".\output\$Output\dumpsys"
    if ($Packages.Count -gt 0)
    {
        foreach ($p in $Packages)
```

```
        {
            $name = "$Service-$p.txt"
            $fullPath = "$path\$name"
            CreateFileIfNotExists -Path $path -Name $name -Type "file"
            Get-Date | Out-File -FilePath "$path\$name" -Append
            adb shell dumpsys $Service $p $OptionalParams | Out-File -FilePath
$fullPath -Append
        }
    }
    else
    {
        $name = "$Service.txt"
        $fullPath = "$path\$name"
        CreateFileIfNotExists -Path $path -Name $name -Type "file"
        Get-Date | Out-File -FilePath $fullPath -Append
        adb shell dumpsys $Service $OptionalParams | Out-File -FilePath $fullPath -
Append
    }
}

function LogCatMonitor
{
    Param([String] $Output)
    Write-Host "LogCat call"
    $path = ".\output\$Output\logcat\logcat.txt"
    Get-Date | Out-File -FilePath $path -Append
    adb logcat -d -v monotonic | Out-File -FilePath $path -Append
    adb logcat -c
}

function ProcTasksMonitor {
    Param([String] $Output)
    Write-Host "ProcTasks call"
    $path = ".\output\$Output\proctasks\proctasks.txt"
    Get-Date | Out-File -FilePath $path -Append
    $dirs_str = adb shell ls /proc/
    $dirs_str = $dirs_str.Split(' ')
    $dirs_str = @($dirs_str) -match '^\d+$'
    $dirs_str_full = $dirs_str | ForEach-Object { "/proc/" + $_ + "/stat" }
    $res = adb shell cat ($dirs_str_full -join ' ')
    if ($null -ne $res) {
        $res | Out-File -FilePath $path -Append
```

```
    }
}

function CreateFileIfNotExists
{
    param (
        $Path,
        $Name,
        $Type
    )
    if (!(Test-Path "$Path\$Name"))
    {
        New-Item -Path $Path -Name $Name -ItemType $Type -Force
    }
}
```

Назва файлу: workload.ps1.

Короткий опис: функції генерування робочого навантаження для ОС Android.

Вихідний код:

```
function RunMonkey
{
    param (
        [String[]] $Packages,
        [Int32] $DurationMs = 0,
        [Int32] $EventsCount = 100,
        [Boolean] $IgnoreErrors = $true,
        [Boolean] $EnableEvents = $true,
        [Boolean] $EnableSwitches = $true
    )
    [Int32] $eventsDelayMs = $DurationMs / $EventsCount
    $packagesParams = ""
    if ($Packages.Count -gt 0)
    {
        $packagesParams = "-p " + [String]::Join(" -p ", $Packages)
    }
    $throttleParam = ""
    if ($eventsDelayMs -gt 0)
    {
        $throttleParam = "--throttle $eventsDelayMs"
    }

    $ignoreParams = ""
```

```
    if ($IgnoreErrors)
    {
        $ignoreParams = "--ignore-crashes --ignore-timeouts --ignore-security-exceptions --kill-process-after-error"
    }
    $eventsParams = ""
    if ($EnableEvents -and $EnableSwitches)
    {
        $eventsParams = "--pct-touch 20 --pct-motion 20 --pct-trackball 15 --pct-nav 20 --pct-majornav 15 --pct-syskeys 0 --pct-appswitch 6 --pct-anyevent 0 --pct-flip 2 --pct-pinchzoom 2"
    }
    elseif ($EnableEvents)
    {
        $eventsParams = "--pct-touch 20 --pct-motion 15 --pct-trackball 15 --pct-nav 20 --pct-majornav 15 --pct-syskeys 5 --pct-anyevent 5 --pct-flip 2 --pct-pinchzoom 3"
    }
    elseif ($EnableSwitches)
    {
        $eventsParams = "--pct-touch 0 --pct-motion 0 --pct-trackball 0 --pct-nav 0 --pct-majornav 0 --pct-syskeys 0 --pct-appswitch 100 --pct-anyevent 0 --pct-flip 0 --pct-pinchzoom 0"
    }
    Write-Host "[Workload Generator] all params: $packagesParams -v -v $throttleParam $eventsParams $ignoreParams $EventsCount"
    adb shell monkey $packagesParams -v -v $throttleParam $eventsParams $ignoreParams $EventsCount
}

function RunPackages
{
    param (
        [String[]] $Packages
    )
    foreach ($package in $Packages)
    {
        adb shell monkey -p $package -c android.intent.category.LAUNCHER 1
    }
}

function KillPackages
{
```

```
    param (
        [String[]] $Packages
    )
    foreach ($package in $Packages)
    {
        #adb shell killall -9 $package
        adb shell am force-stop $package
    }
}
```

Назва файлу: stress.ps1.

Короткий опис: алгоритм виконання стресового тестування, який контролює виконання генерації робочого навантаження та збір системних даних.

Вихідний код:

```
function RunStressTest {
    param (
        [Int32] $IntervalsCount,
        [String] $Cleaner,
        [String] $WorkloadGenerator,
        [String] $Monitor,
        [Int32] $WorkloadDurationTicks = 0,
        [Int32] $PauseDurationSeconds = 0,
        [String] $OnTimeAction = "",
        [Int32] $ActionTime = 0
    )
    & $Cleaner
    $i = 0
    while ($i -ne $IntervalsCount) {
        Write-Host "Monitor iteration  #$i"
        if (($i -gt 0) -and ($ActionTime -gt 0) -and ($OnTimeAction.Length -gt 0)) {
            $j = $i % $ActionTime
            if ($j -eq 0) {
                & $OnTimeAction
            }
        }
        if (($i -gt 0) -and ($WorkloadDurationTicks -gt 0) -and ($PauseDurationSeconds
-gt 0)) {
            if (($i % $WorkloadDurationTicks) -eq 0) {
                Write-Host "Pause begin for $PauseDurationSeconds"
                $sec = 0
                while ($sec -lt $PauseDurationSeconds) {
                    Write-Host "Pause sec = $sec"
```

```
                    Start-Sleep -s 25
                    & $Monitor
                    $sec = $sec + 30#TODO: fix pause timing - remove hardcoded 20/30
                }
            } else {
                & $WorkloadGenerator
            }
        } else {
            & $WorkloadGenerator
        }
        & $Monitor
        $i = $i + 1
    }
}
```

Назва файлу: frames_draw_and_launch_time_comparison_test.ps1.

Короткий опис: налаштування коду для виконання стресового тестування в межах експериментального дослідження EXP1. Порівняння метрик продуктивності GUI із примусовими перезавантаженнями та без перезавантажень.

Вихідний код:

```
. Current-Location\..\scripts\stress.ps1
. Current-Location\..\scripts\workload.ps1
. Current-Location\..\scripts\monitor.ps1

$packages = "com.google.android.youtube", "com.android.contacts",
"com.android.chrome", "com.android.camera", "com.google.android.apps.photos"
$intervalsCount1 = 400
$intervalsCount2 = 500
$intervalDurationMs = 30 * 1000
$intervalEventsCount1 = 80
$intervalEventsCount2 = 100
$output1 = "user_perceived_response_metrics_with_forced_restarts_test"
$output2 = "user_perceived_response_metrics_test"

function RunCleaners
{
    adb logcat -c
    foreach ($p in $Packages)
    {
        adb shell dumpsys gfxinfo $p reset
    }
```

```
}

function RunWorkloadGenerators1
{
    KillPackages -Packages $packages
    RunPackages -Packages $packages
    RunMonkey -Packages $packages -DurationMs $intervalDurationMs -EventsCount $intervalEventsCount1
}

function RunWorkloadGenerators2
{
    RunMonkey -Packages $packages -DurationMs $intervalDurationMs -EventsCount $intervalEventsCount2
}

function RunMonitors1
{
    DumpsysServiceMonitor -Output $output1 -Packages $packages -Service gfxinfo -OptionalParams "framestats reset"
    DumpsysServiceMonitor -Output $output1 -Service meminfo -OptionalParams "-c"
    DumpsysServiceMonitor -Output $output1 -Service graphicsstats -OptionalParams framestats
    LogCatMonitor -Output $output1 -OptionalParams "*:I"
}

function RunMonitors2
{
    DumpsysServiceMonitor -Output $output2 -Packages $packages -Service gfxinfo -OptionalParams "framestats reset"
    DumpsysServiceMonitor -Output $output2 -Service meminfo -OptionalParams "-c"
    DumpsysServiceMonitor -Output $output2 -Service graphicsstats -OptionalParams framestats
    LogCatMonitor -Output $output2 -OptionalParams "*:I"
}

InitOutputDirs -Output $output1 -Packages $packages
InitOutputDirs -Output $output2 -Packages $packages

#RunStressTest -IntervalsCount $intervalsCount1 -Cleaner 'RunCleaners' -WorkloadGenerator 'RunWorkloadGenerators1' -Monitor 'RunMonitors1'
```

```
RunStressTest -IntervalsCount $intervalsCount2 -Cleaner 'RunCleaners' -WorkloadGenerator 'RunWorkloadGenerators2' -Monitor 'RunMonitors2'
```

Назва файлу: native_cross_comparison_test.ps1.

Короткий опис: налаштування коду для виконання стресового тестування в межах експериментального дослідження EXP2. Порівняння native та cross-platform додатків, а також, порівняння сценаріїв використання мобільного пристрою.

Вихідний код:

```
. Current-Location\..\scripts\stress.ps1
. Current-Location\..\scripts\workload.ps1
. Current-Location\..\scripts\monitor.ps1

function RunCleaners {
    param ([String[]] $Packages)
    adb logcat -c
    foreach ($p in $Packages) {
        adb shell dumpsys gfxinfo $p reset
    }
}

function RestartPackages {
    param ([String[]] $Packages)
    KillPackages -Packages $Packages
    RunPackages -Packages $Packages
}

function RunWorkloadGenerators {
    param ([String[]] $Packages)
    RunMonkey -Packages $Packages -DurationMs 60000 -EventsCount 200
}

function RunMonitors {
    param (
        [String] $Output,
        [String[]] $Packages
    )
    DumpsysServiceMonitor -Output $Output -Packages $Packages -Service gfxinfo -OptionalParams "framestats reset"
    DumpsysServiceMonitor -Output $Output -Service meminfo -OptionalParams "-c"
    DumpsysServiceMonitor -Output $Output -Service graphicsstats -OptionalParams framestats
```

```
    LogCatMonitor -Output $Output -OptionalParams "*:I"
    ProcTasksMonitor -Output $Output
}

### Native

$packagesNative = "com.google.android.youtube", "com.android.chrome",
"com.android.camera", "com.google.android.apps.photos", "com.android.contacts"
$outputNativeMultipleApps = "native_multiple_apps_test"
$outputNativeUsageDelays = "native_usage_delays_test"
function RestartNativeApps { RestartPackages -Packages $packagesNative }
function RunNativeCleaners { RunCleaners -Packages $packagesNative }
function RunNativeWorkload { RunWorkloadGenerators -Packages $packagesNative }
function RunNativeMultipleAppsMonitor { RunMonitors -Output $outputNativeMultipleApps
-Packages $packagesNative }
function RunNativeUsageDelaysMonitor { RunMonitors -Output $outputNativeUsageDelays -
Packages $packagesNative }
function RunNativeMultipleAppsTest {
    InitOutputDirs -Output $outputNativeMultipleApps -Packages $packagesNative
    RestartPackages -Packages $packagesNative
    RunStressTest -IntervalsCount 600 -Cleaner 'RunNativeCleaners' -WorkloadGenerator
'RunNativeWorkload' -Monitor 'RunNativeMultipleAppsMonitor' -ActionTime 200 -
OnTimeAction 'RestartNativeApps'
}
function RunNativeUsageDelaysTest {
    InitOutputDirs -Output $outputNativeMultipleApps -Packages $packagesNativeApps
    RestartPackages -Packages $packagesNativeApps
    RunStressTest -IntervalsCount 400 -Cleaner 'RunNativeCleaners' -WorkloadGenerator
'RunNativeWorkload' -Monitor 'RunNativeUsageDelaysMonitor' -ActionTime 150 -
OnTimeAction 'RestartNativeApps' -WorkloadDurationTicks 100 -PauseDurationSeconds 1800
}

### Flutter

$packagesFlutter = "com.reflectlyApp", "com.hamilton.app", "com.ebay.motorsapp",
"com.spotlightsix.zentimerlite2", "com.mgmresorts.mgmresorts"#,"com.grab.merchant",
"com.google.stadia.android"
$outputFlutterMultipleApps = "flutter_multiple_apps_test"
$outputFlutterUsageDelays = "flutter_usage_delays_test"
function RestartFlutterApps { RestartPackages -Packages $packagesFlutter }
function RunFlutterCleaners { RunCleaners -Packages $packagesFlutter }
function RunFlutterWorkload { RunWorkloadGenerators -Packages $packagesFlutter }
```

```
function RunFlutterMultipleAppsMonitor { RunMonitors -Output $outputFlutterMultipleApps -Packages $packagesFlutter }
function RunFlutterUsageDelaysMonitor { RunMonitors -Output $outputFlutterUsageDelays -Packages $packagesFlutter }
function RunFlutterMultipleAppsTest {
    InitOutputDirs -Output $outputFlutterMultipleApps -Packages $packagesFlutter
    RestartPackages -Packages $packagesFlutter
    RunStressTest -IntervalsCount 600 -Cleaner 'RunFlutterCleaners' -WorkloadGenerator 'RunFlutterWorkload' -Monitor 'RunFlutterMultipleAppsMonitor' -ActionTime 200 -OnTimeAction 'RestartFlutterApps'
}
function RunFlutterUsageDelaysTest {
    InitOutputDirs -Output $outputFlutterUsageDelays -Packages $packagesFlutter
    RestartPackages -Packages $packagesFlutter
    RunStressTest -IntervalsCount 400 -Cleaner 'RunFlutterCleaners' -WorkloadGenerator 'RunFlutterWorkload' -Monitor 'RunFlutterUsageDelaysMonitor' -ActionTime 150 -OnTimeAction 'RestartFlutterApps' -WorkloadDurationTicks 100 -PauseDurationSeconds 1800
}

### React Native

$outputReactMultipleApps = "react_multiple_apps_test"
$packagesReactMultipleApps = "com.shinetext.shine", "com.pinterest", "com.cleevio.spendee", "com.myntra.android"#, "com.edutapps.maphi"

### Entry point

Write-Host "Run test:
    - native   multiple apps  (1)
    - native   usage delay    (11)
    - flutter  multiple apps  (2)
    - flutter  usage delay    (22)"
$run_code = Read-Host -Prompt 'Input run code:'
switch ($run_code) {
    1 { RunNativeMultipleAppsTest; Break }
    11 { RunNativeUsageDelaysTest; Break }
    2 { RunFlutterMultipleAppsTest; Break }
    22 { RunFlutterUsageDelaysTest; Break }
    Default { Write-Host "Ooops! :)" }
}
```

Додаток Г Лістинг модуля моделей старіння та омолодження

Назва файлу: RungeKuttaMethod.kt.

Короткий опис: програмна реалізація методу Рунге-Кутта 4-го порядку.

Вихідний код:

```
class RungeKuttaMethod(
    val coefficients: Array<Array<Double>>
) : Method {

    override fun solve(initialX: Double, deltaX: Double, maxX: Double, initialY:
List<Double>): List<List<Double>> {
        val result: ArrayList<List<Double>> = ArrayList()
        val maxTime: Int = ((maxX - initialX) / deltaX).toInt()
        val equationCount = coefficients.size
        val step = deltaX
        val initialValues = initialY.toDoubleArray()
        val k0 = DoubleArray(maxTime * equationCount)
        val k1 = DoubleArray(maxTime * equationCount)
        val k1_help = DoubleArray(equationCount)
        val k2 = DoubleArray(maxTime * equationCount)
        val k2_help = DoubleArray(equationCount)
        val k3 = DoubleArray(maxTime * equationCount)
        val k3_help = DoubleArray(equationCount)
        val res = DoubleArray(maxTime * equationCount)
        var k = 0
        while (k < maxTime) {
            val resultTmpl: ArrayList<Double> = ArrayList()
            for (i in 0 until equationCount) {
                k0[i + k * equationCount] = func(initialValues, i, equationCount,
coefficients) * step
            }
            for (i in 0 until equationCount) {
                k1_help[i] = initialValues[i] + k0[i + k * equationCount] / 2
            }
            for (i in 0 until equationCount) {
                k1[i + k * equationCount] = func(k1_help, i, equationCount,
coefficients) * step
            }
            for (i in 0 until equationCount) {
                k2_help[i] = initialValues[i] + k1[i + k * equationCount] / 2
            }
            for (i in 0 until equationCount) {
```

```
                k2[i + k * equationCount] = func(k2_help, i, equationCount,
coefficients) * step
            }
            for (i in 0 until equationCount) {
                k3_help[i] = initialValues[i] + k2[i + k * equationCount]
            }
            for (i in 0 until equationCount) {
                k3[i + k * equationCount] = func(k3_help, i, equationCount,
coefficients) * step
            }
            for (i in 0 until equationCount) {
                res[i + k * equationCount] =
                    initialValues[i] + 1.0 / 6.0 * (k0[i + k * equationCount] + 2 *
k1[i + k * equationCount] + 2 * k2[i + k * equationCount] + k3[i + k * equationCount])
            }
            for (i in 0 until equationCount) {
                initialValues[i] = res[i + k * equationCount]
                resultTmpl.add(res[i + k * equationCount])
            }
            result.add(resultTmpl)
            k += 1
        }
        return result
    }

    private fun func(arr: DoubleArray, i: Int, n: Int, func: Array<Array<Double>>):
Double {
        var res = 0.0
        for (j in 0 until n) res += arr[j] * func[i][j]
        return res
    }

}
```

Назва файлу: Model.kt.

Короткий опис: базовий клас опису аналітичної моделі та симуляції процесу.

Вихідний код:

```
abstract class Model() {
    private var initialTime: Double = 0.0
    private var intervalTime: Double = 1.0
    private var maxTime: Double = 1000.0
    private var result: List<List<Double>> = listOf()
```

```
    abstract fun modelName(): String
    abstract fun namesOfStates(): List<String>
    abstract fun initialProbabilities(): List<Double>
    abstract fun method(): Method
    fun solve(initialTime: Double, intervalTime: Double, maxTime: Double):
List<List<Double>> {
        this.initialTime = initialTime
        this.initialTime = intervalTime
        this.maxTime = maxTime
        val initialProbabilities = initialProbabilities()
        result = method().solve(initialTime, intervalTime, maxTime,
initialProbabilities)
        return result
    }

    fun save() {
        if (result.isNotEmpty()) {
            val fullResults = fullResults(result)
            val columns = arrayListOf("timestamp").apply {
                addAll(namesOfStates())
                add("sum")
            }
            val filePath = "results/${modelName()}.csv"
            fullResults.saveToCsv(filePath = filePath, columns = columns)
        }
    }

    private fun fullResults(data: List<List<Double>>): List<List<Double>> {
        return data.mapIndexed { index, list -> arrayListOf<Double>().apply {
            add(initialTime + (index * intervalTime))
            addAll(list)
            add(list.sum())
        } }.toList()
    }

}
```

Назва файлу: ModelTest.kt.

Короткий опис: клас-симулятор процесу старіння для різних значень інтенсивності виконання омолодження.

Вихідний код:

```
class ModelTest(
```

```
    private val initialTime: Double,
    private val maxTime: Double,
    private val intervalTime: Double
) {
    fun test(
        model: (lambdaYR: Double) -> Model
    ): ModelTestResults {
        val results: ArrayList<List<Double>> = ArrayList()
        val modelName = model(0.0).modelName()
        val namesOfStates = model(0.0).namesOfStates()
        var t = initialTime + intervalTime
        while (t < maxTime) {
            println("$modelName test iteration #${(t / intervalTime) + 1}/${(maxTime
- initialTime) / intervalTime}")
            val lambdaYR = 1.0 / t
            val result: ArrayList<Double> = ArrayList()
            result.add(lambdaYR)
            result.addAll(model(lambdaYR).solve(initialTime, intervalTime, t).last())
            results.add(result)
            t += intervalTime
        }
        return ModelTestResults(
            initialTime = initialTime,
            intervalTime = intervalTime,
            maxTime = maxTime,
            modelName = modelName,
            namesOfStates = namesOfStates,
            result = results
        )
    }
}
```

Назва файлу: EnhancedSoftwareAgingRejuvenationModel.kt.

Короткий опис: модель старіння та омолодження із різними рівнями старіння.

Вихідний код:

```
class EnhancedSoftwareAgingRejuvenationModel(
    val lamYA: Double,
    val lamAO: Double,
    val lamOAf: Double,
    val lamAR: Double,
    val lamRY: Double,
    val lamOR: Double,
```

```
    val lamAfY: Double,
    val initialY: Double,
    val initialA: Double,
    val initialO: Double,
    val initialAf: Double,
    val initialR: Double
) : Model() {

    override fun method(): Method {
        return RungeKuttaMethod (
            arrayOf(
                arrayOf(lamYA * -1, 0.0, 0.0, lamAfY, lamRY),// S
                arrayOf(lamYA, (lamAO + lamAR) * -1, 0.0, 0.0, 0.0),// A
                arrayOf(0.0, lamAO, (lamOR + lamOAf) * -1, 0.0, 0.0),// O
                arrayOf(0.0, 0.0, lamOAf, lamAfY * -1, 0.0),// Af
                arrayOf(0.0, lamAR, lamOR, 0.0, lamRY * -1)// R
            )
        )
    }

    override fun initialProbabilities(): List<Double> {
        return listOf(
            initialY, // 0
            initialA, // 1
            initialO, // 2
            initialAf, // 3
            initialR // 4
        )
    }

    override fun namesOfStates(): List<String> {
        return listOf(
            "Y", // 0
            "A", // 1
            "O", // 2
            "Af", // 3
            "R" // 4
        )
    }

    override fun modelName(): String = "EnhancedSoftwareAgingRejuvenationModel"
```

```
}
```

Назва файлу: MobileDeviceWithRejuvenationModel.kt.

Короткий опис: модель старіння та омолодження з урахуванням рівня заряду батареї мобільного пристрою.

Вихідний код:

```
class MobileDeviceWithRejuvenationModel(
    val lamSA: Double,
    val lamAS: Double,
    val lamYO: Double,
    val lamORe: Double,
    val lamReY: Double,
    val lamYR: Double,
    val lamRY: Double,
    val lamSpLp: Double,
    val lamLpSp: Double,
    val lamLpOp: Double,
    val initialSYSp: Double,
    val initialAYSp: Double,
    val initialSOSp: Double,
    val initialAOSp: Double,
    val initialRASp: Double,
    val initialRSSp: Double,
    val initialAReSp: Double,
    val initialSReSp: Double,
    val initialSYLp: Double,
    val initialAYLp: Double,
    val initialSOLp: Double,
    val initialAOLp: Double,
    val initialRALp: Double,
    val initialRSLp: Double,
    val initialAReLp: Double,
    val initialSReLp: Double,
    val initialOp: Double
) : Model() {

    override fun method(): Method {
        return RungeKuttaMethod (
            arrayOf(
                arrayOf((lamSA + lamYO + lamYR + lamSpLp) * -1, lamAS, 0.0, 0.0, 0.0,
lamRY, 0.0, lamReY, lamLpSp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),// SYSp
```

```
                arrayOf(lamSA, (lamAS + lamYO + lamSpLp) * -1, 0.0, 0.0, lamRY, 0.0,
lamReY, 0.0, 0.0, lamLpSp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),// AYSp
                arrayOf(lamYO, 0.0, (lamORe + lamSA + lamSpLp) * -1, lamAS, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0),// SOSp
                arrayOf(0.0, lamYO, lamSA, (lamORe + lamAS + lamSpLp) * -1, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0, 0.0, 0.0, 0.0, 0.0),// AOSp
                arrayOf(0.0, 0.0, 0.0, 0.0, (lamRY + lamAS + lamSpLp) * -1, lamSA,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0, 0.0, 0.0, 0.0),// RASp
                arrayOf(lamYR, 0.0, 0.0, 0.0, lamAS, (lamRY + lamSA + lamSpLp) * -1,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0, 0.0, 0.0),// RSSp
                arrayOf(0.0, 0.0, 0.0, lamORe, 0.0, 0.0, (lamReY + lamAS + lamSpLp) *
-1, lamSA, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0, 0.0),// AReSp
                arrayOf(0.0, 0.0, lamORe, 0.0, 0.0, 0.0, lamAS, (lamReY + lamSA +
lamSpLp) * -1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamLpSp, 0.0),// SReSp

                arrayOf(lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, (lamSA + lamYO +
lamLpSp + lamLpOp) * -1, lamAS, 0.0, 0.0, 0.0, lamRY, 0.0, lamReY, 0.0),// SYLp
                arrayOf(0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamSA, (lamAS +
lamYO + lamLpSp + lamLpOp) * -1, 0.0, 0.0, lamRY, 0.0, lamReY, 0.0, 0.0),// AYLp
                arrayOf(0.0, 0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0, lamYO, 0.0, (lamORe
+ lamSA + lamLpSp + lamLpOp) * -1, lamAS, 0.0, 0.0, 0.0, 0.0, 0.0),// SOLp
                arrayOf(0.0, 0.0, 0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0, lamYO, lamSA,
(lamORe + lamAS + lamLpSp + lamLpOp) * -1, 0.0, 0.0, 0.0, 0.0, 0.0),// AOLp
                arrayOf(0.0, 0.0, 0.0, 0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, (lamRY + lamAS + lamLpSp + lamLpOp) * -1, lamSA, 0.0, 0.0, 0.0),// RALp
                arrayOf(0.0, 0.0, 0.0, 0.0, 0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, lamAS, (lamRY + lamSA + lamLpSp + lamLpOp) * -1, 0.0, 0.0, 0.0),// RSLp
                arrayOf(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamSpLp, 0.0, 0.0, 0.0, 0.0,
lamORe, 0.0, 0.0, (lamReY + lamAS + lamLpSp + lamLpOp) * -1, lamSA, 0.0),// AReLp
                arrayOf(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamSpLp, 0.0, 0.0, lamORe,
0.0, 0.0, 0.0, lamAS, (lamReY + lamSA + lamLpSp + lamLpOp) * -1, 0.0),// SReLp

                arrayOf(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, lamLpOp, lamLpOp,
lamLpOp, lamLpOp, lamLpOp, lamLpOp, lamLpOp, lamLpOp, 0.0)// Op
            )
        )
    }

    override fun initialProbabilities(): List<Double> {
        return listOf(
            initialSYSp, // 0
            initialAYSp, // 1
```

```
            initialSOSp, // 2
            initialAOSp, // 3
            initialRASp, // 4
            initialRSSp, // 5
            initialAReSp, // 6
            initialSReSp, // 7
            initialSYLp, // 8
            initialAYLp, // 9
            initialSOLp, // 10
            initialAOLp, // 11
            initialRALp, // 12
            initialRSLp, // 13
            initialAReLp, // 14
            initialSReLp, // 15
            initialOp // 16
        )
    }

    override fun namesOfStates(): List<String> {
        return listOf(
            "SYSp", // 0
            "AYSp", // 1
            "SOSp", // 2
            "AOSp", // 3
            "RASp", // 4
            "RSSp", // 5
            "AReSp", // 6
            "SReSp", // 7
            "SYLp", // 8
            "AYLp", // 9
            "SOLp", // 10
            "AOLp", // 11
            "RALp", // 12
            "RSLp", // 13
            "AReLp", // 14
            "SReLp", // 15
            "Op" // 16
        )
    }

    override fun modelName(): String = "MobileDeviceWithRejuvenationModel"
}
```

Додаток Ґ Список публікацій здобувача за темою дисертації та відомості про апробацію результатів дисертації

Наукові праці, в яких опубліковано основні наукові результати дисертації

1. Яковина В.С., Угриновський Б.В. Старіння програмного забезпечення в контексті його надійності: огляд проблематики. Науковий вісник НЛТУ України. 2019. 29(5), 123-128. DOI: https://doi.org/10.15421/40290525

2. Яковина В.С., Угриновський Б.В. Старіння програмного забезпечення мобільних додатків: огляд проблематики. Науковий вісник НЛТУ України. 2020. 30(2), 107-112. DOI: https://doi.org/10.36930/40300219

3. Яковина В.С., Угриновський Б.В. Метрики інтерфейсу користувача для виявлення явища старіння програмного забезпечення в мобільній системі Android. Вісник НУЛП «Інформаційні системи та мережі». 2021. 9, 32-43. DOI: https://doi.org/10.23939/sisn2021.09.032

4. Яковина В.С., Угриновський Б.В. Дослідження системних процесів та користувацьких додатків операційної системи Android в контексті старіння програмного забезпечення. Вісник ХНУ: Технічні науки. 2021. 295(2), 64-70. DOI: https://www.doi.org/10.31891/2307-5732-2021-295-2-64-70

5. Yakovyna, V. S., Uhrynovskyi, B. V. Android software aging and rejuvenation model considering the battery charge. Radio Electronics, Computer Science, Control. 2021. (4), 140-148. DOI: https://doi.org/10.15588/1607-3274-2021-4-13

6. Yakovyna V. S., Uhrynovskyi B. V. Extended software aging and rejuvenation model for Android operating system considering different aging levels and rejuvenation procedure types. Computer systems and information technologies. 2021. 3, 116-124. DOI: https://doi.org/10.31891/CSIT-2021-5-9

7. Яковина В.С., Угриновський Б.В. Метод омолодження програмного забезпечення для операційної системи Android з використанням комплексної моделі його старіння на підставі ланцюга Маркова. Науковий вісник НЛТУ України. 2021. 31(6), 97-103. DOI: https://doi.org/10.36930/40310615

Праці, які засвідчують апробацію матеріалів дисертації

1. Yakovyna V. S., Uhrynovskyi B. V., Bachkay O. Software Failures Forecasting by Holt - Winters, ARIMA and NNAR Methods. *IEEE 14th International Conference on Computer Sciences and Information Technologies*. 2019. DOI: https://doi.org/10.1109/STC-CSIT.2019.8929863

2. Yakovyna V. S., Uhrynovskyi B. V. User-Perceived Response Metrics in Android OS for Software Aging detection. *IEEE 15th International Conference on Computer Sciences and Information Technologies*. 2020. DOI: https://doi.org/10.1109/CSIT49958.2020.9322031

3. Яковина В.С., Угриновський Б.В. Модель старіння та омолодження програмного забезпечення для платформи Android. *XX Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2020. 5-8 червня.

4. Яковина В.С., Угриновський Б.В. Засоби протидії явищу старіння програмного забезпечення в операційній системі Android. *XXI Міжнародна науково-технічна конференція «Вимірювальна та обчислювальна техніка в технологічних процесах»*. 2021. 3-7 червня.

5. Yakovyna, V. S., Uhrynovskyi, B. V. Aging of Native and Flutter Applications in Android OS in Various Usage Scenarios. *IEEE 16th International Conference on Computer Sciences and Information Technologies*. 2021.

Додаток Д Акти впровадження результатів дисертації

ЗАТВЕРДЖУЮ
Проректор з науково-педагогічної роботи
НУ «Львівська політехніка»
_____ к.т.н., доц. Давидчак О. Р.
“15” ____ 2022 р.

АКТ
про впровадження результатів дисертаційної роботи
Угриновського Богдана Володимировича
«Методи і засоби підвищення надійності програмного забезпечення з урахуванням процесу його старіння», представленої на здобуття наукового ступеня доктора філософії за спеціальністю 121 «Інженерія програмного забезпечення»,
у навчальному процесі кафедри програмного забезпечення

Даний акт складений комісією у складі:
д.т.н., проф. Федасюк Д. В. – завідувач кафедри програмного забезпечення;
д.т.н., проф. Журавчак Л. М. – професор кафедри програмного забезпечення;
к.т.н., доц. Сенів М. М. – доцент кафедри програмного забезпечення, лектор дисципліни «Теорія надійності програмних систем».

Цим актом підтверджується використання результатів дисертаційної роботи Угриновського Богдана Володимировича на тему «Методи і засоби підвищення надійності програмного забезпечення з урахуванням процесу його старіння» в навчальному процесі кафедри програмного забезпечення для студентів спеціальності 121 «Інженерія програмного забезпечення» в лекційному курсі та практикумі дисципліни освітньо-кваліфікаційного рівня магістр «Теорія надійності програмних систем».

Розроблені у дисертації Марковські моделі процесу старіння та омолодження програмного забезпечення можуть бути застосовані для оцінки показників якості існуючого програмного забезпечення, зокрема, для оцінки надійності. Вивчення та аналіз процесу старіння дає змогу розробляти відповідні засоби для протидії старінню, що в свою чергу дозволяє підвищити надійність програмного забезпечення, зокрема, шляхом розроблення методів його омолодження.

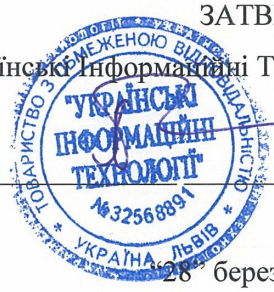
Члени комісії:
зав. каф. програмного забезпечення,
д.т.н., професор _____ Федасюк Д. В.

проф. каф. програмного забезпечення,
д.т.н., професор _____ Журавчак Л. М.

доц. каф. програмного забезпечення,
к.т.н., доцент, _____ Сенів М. М.

ЗАТВЕРДЖУЮ

Директор ТОВ «Українські Інформаційні Технології»



_____

«28» березня 2022 р.

АКТ

про дослідне випробування результатів

дисертаційного дослідження аспіранта кафедри програмного забезпечення

Національного університету «Львівська політехніка»

Угриновського Богдана Володимировича

Даний акт складений про те, що середовище експериментальних досліджень явища старіння програмного забезпечення в користувацьких застосуваннях операційної системи Android, розроблене під час виконання дисертаційної роботи Угриновського Богдана Володимировича на тему «Методи і засоби підвищення надійності програмного забезпечення з урахуванням процесу його старіння», пройшло дослідне випробування на ТОВ «Українські Інформаційні Технології» (м. Львів).

Дане середовище дозволяє визначати такі ефекти процесу старіння як погіршення продуктивності графічного інтерфейсу користувача та збільшення використання ресурсів пам'яті в розроблюваних мобільних застосування з урахуванням інтенсивності генерування робочого навантаження та сценарію використання мобільного пристрою.

Використання розробленого середовища дає можливість виявляти вразливі до старіння застосування та сценарії використання. Зокрема, процеси старіння спостерігались приблизно в 25% виконаних тестів. Виявлені процеси старіння та умови старіння на етапі тестування дозволяють приймати рішення про необхідність покращення та виправлення існуючих проблем продуктивності та надійності під час розроблення програмного забезпечення.

Даний акт не є підставою для взаємних фінансових розрахунків.

ЗАТВЕРДЖУЮ
Директор ПП LinkUP Studio

"02" квітня 2022 р.

АКТ

про дослідне випробування результатів
дисертаційного дослідження аспіранта кафедри програмного забезпечення
Національного університету «Львівська політехніка»
Угриновського Богдана Володимировича

Даний акт складений про те, що комплексна модель старіння та омолодження програмного забезпечення для мобільної системи на основі ланцюга Маркова з неперервним часом розподілу, розроблена під час виконання дисертаційної роботи Угриновського Богдана Володимировича на тему «Методи і засоби підвищення надійності програмного забезпечення з урахуванням процесу його старіння», пройшла дослідне випробування на ПП LinkUP Studio (м. Львів).

Дана модель дозволяє виконувати оцінку показників якості існуючого програмного забезпечення з урахуванням різних рівнів погіршення продуктивності в контексті явища старіння, активності використання мобільного пристрою користувачем та фактору заряду батареї. Модель дозволяє оцінити час до відмови через старіння та ймовірність безвідмовної роботи в той чи інший момент часу. Зокрема, прогнозовану ймовірність безвідмовної роботи програмного забезпечення можна збільшити приблизно в 1,5 рази при умові виконання процедури омолодження.

Використання розробленої моделі дає можливість робити оцінку якості програмного забезпечення та без необхідності проведення реальних експериментальних досліджень які потребують додаткового часу та бюджету.

Даний акт не є підставою для взаємних фінансових розрахунків.